Express.js

옥창원
프론트엔드개발팀 / UIT 개발실
대외비

NHN TECHNOLOGY SERVICES

목차

1. 개요

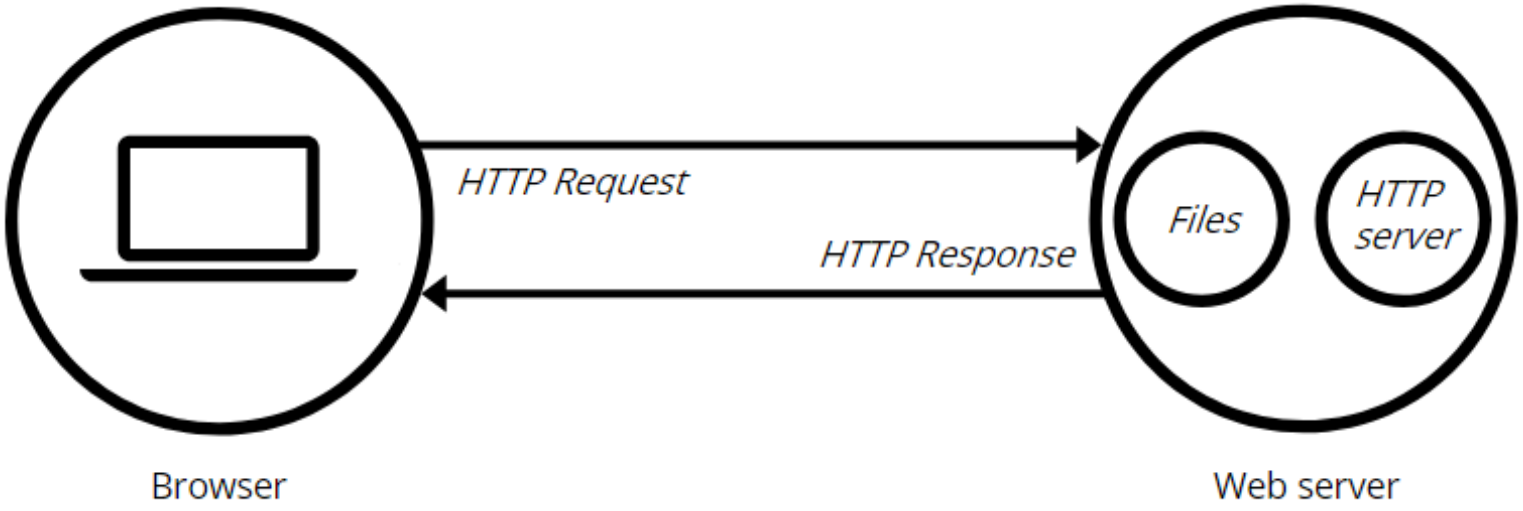
express.js 란?

- Node.js 에서 웹서버를 개발할 때 가장 많이 사용하는 프레임워크 모듈
- 가볍고 간결한 기능을 제공하며 다양한 미들웨어를 연계하여 여러 기능을 구현
- 서버 구동부터 라우팅, 서버사이드 렌더링, Restful API 개발, 쿠키, 세션, 인증 등
- 공식 홈페이지(http://expressjs.com/ko)에 가이드와 레퍼런스 제공

1. 개요

웹서버의 역할

- 웹브라우저가 필요로 하는 컨텐츠를 제공하는 것이 기본적인 역할
- 정적 컨텐츠(html, js, css, img 등의 파일) 또는 동적 컨텐츠(정보, Database) 제공
- HTTP 프로토콜을 통해 요청과 응답을 처리



1. 개요

HTTP 프로토콜

- HTTP(HyperText Transfer Protocol, 문화어: 초본문전송규약, 하이퍼본문전송규약)
- 서버와 클라이언트가 어떻게 메시지를 교환할 지를 정해놓은 규칙
- 80번 포트를 사용하며 요청(Request)과 응답(response)으로 구성
- https://developer.mozilla.org/ko/docs/Web/HTTP/Messages

HTTP의 메서드

- GET, PUT, POST와 같은 동사형 혹은 HEAD, OPTIONS와 같은 명사형이 있으며, 수행되어야 할 동작을 설명.
- 예를 들어, GET은 하나의 리소스를 불러와야 한다는 것을 가리키며, POST는 데이터가 서버로 들어가야 함을 의미.

1. 개요

HTTP 요청 메시지 구조

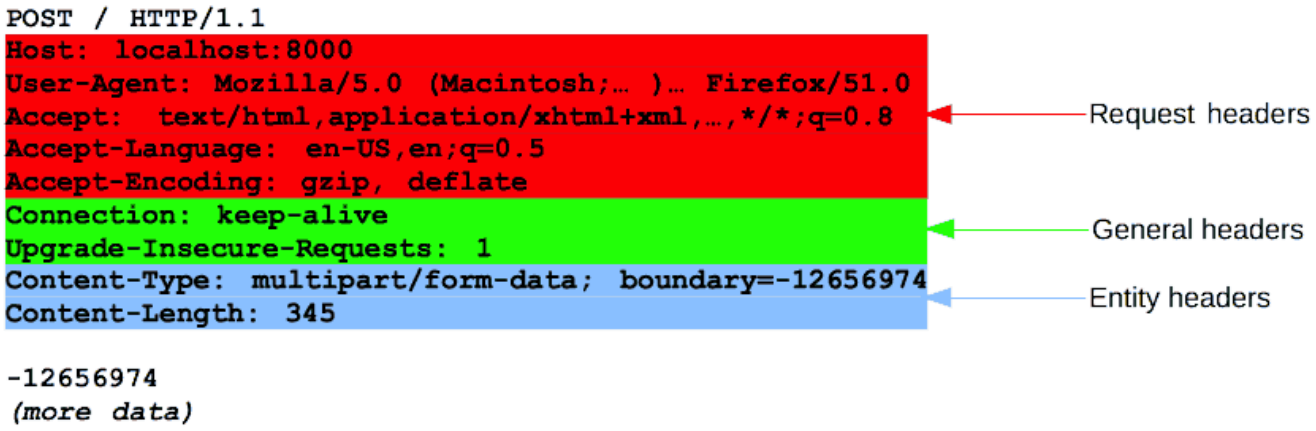
시작줄

GET /img/background.png HTTP/1.1

GET /test.html?query=alibaba HTTP/1.1

POST /registraion HTTP 1.1

헤더



본문

POST 메서드의 경우 전송할 폼 데이터를 본문 영역에 담아 전송

1. 개요

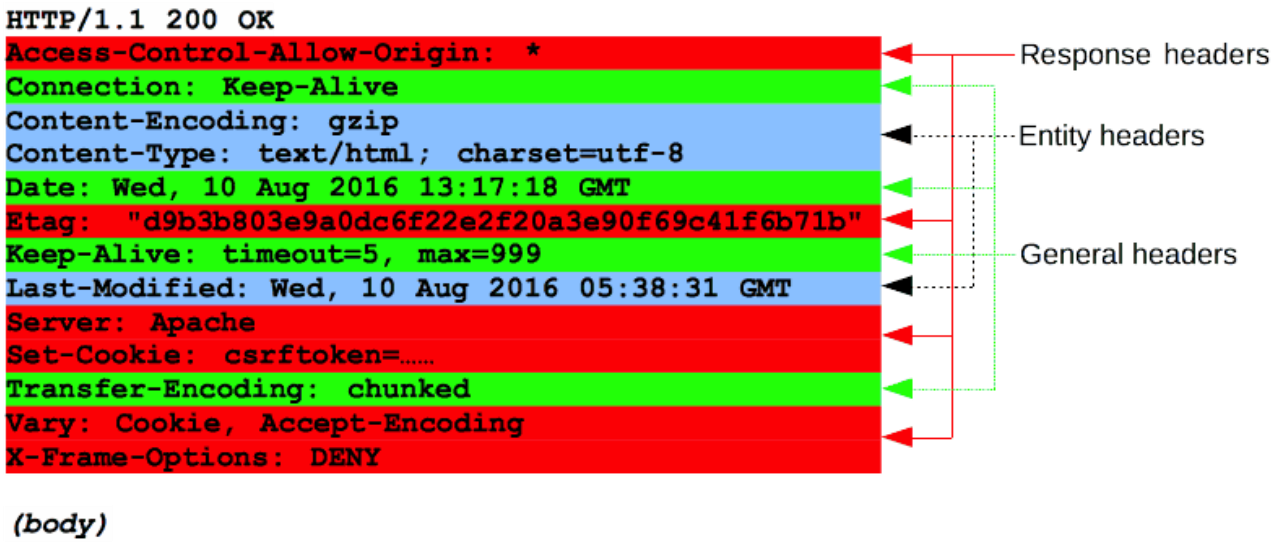
HTTP 응답 메시지 구조

상태줄

HTTP/1.1 200 OK

HTTP/1.1 404 Not Found.

헤더



본문

html, js, css, json, xml 혹은 이미지의 바이너리 데이터 등 컨텐츠 정보를 담는 영역

1. 개요

express.js 설치

```
$ npm install express --save
```

프로젝트 자동 생성

- express-generator 를 사용하면 기본 골격이 갖춰진 프로젝트가 한번에 생성됨

```
$ npm install express-generator -g
$ express myproject --ejs
```

Postman 설치

- 개발된 API를 테스트할 수 있게 해주는 플랫폼
- https://www.getpostman.com/apps 에서 다운로드

2. 기본 동작 구조

간단한 웹서버 작성

basic.server.js

```
// Express 모듈 추출
var express = require('express');

// Express 서버 객체 생성
var app = express();

// HTTP 요청에 대한 응답
app.use(function(request, response){
		response.send('<h1>hello express</h1>');
});

// 서버 구동
app.listen(8080, function(){
		console.log('web server started');
});
```

터미널 실행

```
$ node basic.server.js
web server started
```

- http://localhost:8080 으로 접속하면 hello express를 출력
- 어떤 요청을 보내도 동일한 응답

2. 기본 동작 구조

request 객체

요청에 대한 정보를 제공하는 객체

request 객체의 속성과 메서드

request.query	요청 매개변수를 추출합니다.
request.params	라우팅 매개변수를 추출합니다.
request.headers	요청 헤더를 추출합니다.
request.header()	요청 헤더의 속성을 지정 또는 추출합니다.

response 객체

요청에 대한 응답을 담당하는 객체

response 객체의 메서드

response.send([body])	매개변수의 자료형에 따라 적절한 형태로 응답
response.json([body])	JSON 형태로 응답
response.jsonp([body])	JSONP 형태로 응답
response.redirect([status,] path)	웹 페이지 경로를 강제로 이동

3. 라우팅

라우트 메서드

- HTTP 요청의 메서드와 URL(PATH) 형태에 따라 분기하는 역할
- HTTP 메서드로부터 파생

주요 메서드

get(path, callback[, callback. ..])	GET 요청이 발생했을 때의 이벤트 리스너를 지정
post(path, callback[, callback. ..])	POST 요청이 발생했을 때의 이벤트 리스너를 지정
put(path, callback[, callback. ..])	PUT 요청이 발생했을 때의 이벤트 리스너를 지정
delete(path, callback[, callback. ..])	DELETE 요청이 발생했을 때의 이벤트 리스너를 지정
all(path, callback[, callback. ..])	HTTP 메서드 종류에 상관없이 요청이 발생했을 때의 이벤트 리스너를 지정

이 외에도 다음과 같은 메서드를 지원

get, post, put, head, delete, options, trace, copy, lock, mkcol, move, purge, propfind, proppatch, unlock, report, mkactivity, checkout, merge, m-search, notify, subscribe, unsubscribe, patch, search 및 connect

3. 라우팅

라우트 경로

- 요청 메서드와 조합하여, 요청이 이루어질 수 있는 엔드포인트를 정의
- 라우트 경로는 문자열, 문자열 패턴 또는 정규식으로 정의 가능

문자열

app.get('/home', function....)

문자열 패턴

app.get('/user/:id', function...)

정규식

app.get(/₩.js$/, function...)

3. 라우팅

라우트 예제

route.server.js

```
// Express 모듈 추출
var express = require('express');

// Express 서버 객체 생성
var app = express();

// 라우터 설정
app.get('/a', function(request, response){
                response.send('<a href="/b">Go to B</a>');
});

app.get('/b', function(request, response){
                response.send('<a href="/a">Go to A</a>');
});

// 서버 구동
app.listen(8080, function(){
                console.log('web server started');
});
```

3. 라우팅

request 객체의 query/params 속성

request.query ?name=A와 같은 요청 매개 변수 제공

request.params /:id 처럼 ':' 기호를 사용해 지정된 라우팅 매개 변수 제공

요청 매개변수 query 사용 예

```
app.get('/page', function(request, response){
                response.send(request.query.name + '님 환영합니다.');
});
```

- "http://localhost/page?name=홍길동" 호출

라우팅 매개변수 params 사용 예

```
app.get('/page/:name', function(request, response){
                response.send(request.params.name + '님 환영합니다.');
});
```

- "http://localhost/page/홍길동" 호출

라우터 모듈화

- 규모가 커지면 라우터를 분리할 필요가 생김
- express가 제공하는 express.Router 클래스를 사용하여 모듈로 분리

routerA.js

```
var express = require('express');
var router = express.Router();

router.get('/', function(req, res) {
    res.send('Birds home page');
});

router.get('/about', function(req, res) {
    res.send('About birds');
});

module.exports = router;
```

이후 앱 내에서 다음과 같이 라우터 모듈을 로드

```
app.use('/a', require('./routerA'));
app.use('/b', require('./routerB'));
```

http://localhost/a, http://localhost/a/about 등의 형식으로 페이지 호출 가능

4. 미들웨어

미들웨어 개요

- http 모듈과 express 모듈의 가장 큰 차이는 미들웨어의 유무
- 요청에 대한 응답을 완료하기 전까지 요청 중간에서 여러가지 일을 처리
- use() 메서드로 미들웨어를 적용
- next() 함수를 호출하여 다음 미들웨어로 제어권을 넘김

미들웨어 적용 기본 형태

```
app.use(경로, 미들웨어 함수);
```

- 경로 : 미들웨어가 적용되는 경로. 생략 가능 (생략시 기본값 "/");
- 미들웨어 함수 : function(request, response, next){ } 형태의 함수

URL 경로가 매칭되는 요청에만 미들웨어 적용(하위 경로 포함)

```
app.use("/special", function(request, response, next){
    // 처리로직
    next(); // 동일한 경로에 적용된 다음 미들웨어로 제어를 넘김
});
```

4. 미들웨어

미들웨어 작성

middleware.server.js

```
var express = require('express');
var app = express();

// 스페셜 미들웨어 설정
app.use("/special", function(request, response, next){
                    console.log('스페셜 미들웨어');
                    response.end("special middleware!")
});

// 미들웨어 설정 1
app.use(function(request, response, next){
                    console.log('미들웨어 1');
                    next();
});

// 미들웨어 설정 2
app.use(function(request, response, next){
                    console.log('미들웨어 2');
                    response.end("middleware 1,2!")
});

// 서버 구동
app.listen(8080, function(){
                    console.log('web server started');
});
```

- 서버를 실행하고 페이지를 호출하면 설정한 미들웨어들이 차례대로 실행되어 콘솔에 출력

- 참고) Express 앱에서 사용하기 위한 미들웨어 작성(http://expressjs.com/ko/guide/writing-middleware.html)

4. 미들웨어

static 미들웨어

- express 에서기본 제공하는 미들웨어
- 이미지, CSS, JS, HTML 파일과 같은 정적 리소스를 제공할 루트 경로를 지정하는 역할

기본 호출 형태

```
app.use(express.static(경로));
```

- 입력한 경로 하위에 포함된 파일들을 제공
- 경로/index.html 의 경우 http://localhost/index.html 로 접근
- 경로/img/logo.png 의 경우 http://localhost/img/logo.png 로 접근

가상의 경로를 지정하여 제공

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

- "/public/파일경로"으로 호출 시 "/static/파일경로" 의 파일을 제공
- /static/index.html 의 경우 http://localhost/public/index.html 로 접근
- /static/img/logo.png 의 경우 http://localhost/public/img/logo.png 로 접근

static 미들웨어

static.server.js

```
// Express 모듈 추출
var express = require('express');

// Express 서버 객체 생성
var app = express();

// 정적 리소스 경로 지정
app.use(express.static(__dirname + '/static'));

// 서버 구동
app.listen(8080, function(){
                    console.log('web server started');
});
```

4. 미들웨어

body-parser 미들웨어

- POST 요청시 전달된 데이터를 추출하는 미들웨어
- body-parser 미들웨어를 사용하면 request객체에 body 속성이 부여됨

body-parser 미들웨어 설치

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

사용 예

post.server.js

```
var express = require('express');
var app = express();
var bodyParser = require('body-parser');

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

app.post('/post', function(request, response){
		response.send("POST Data : " + request.body.content);
});

app.listen(8080, function(){
		console.log('web server started');
});
```

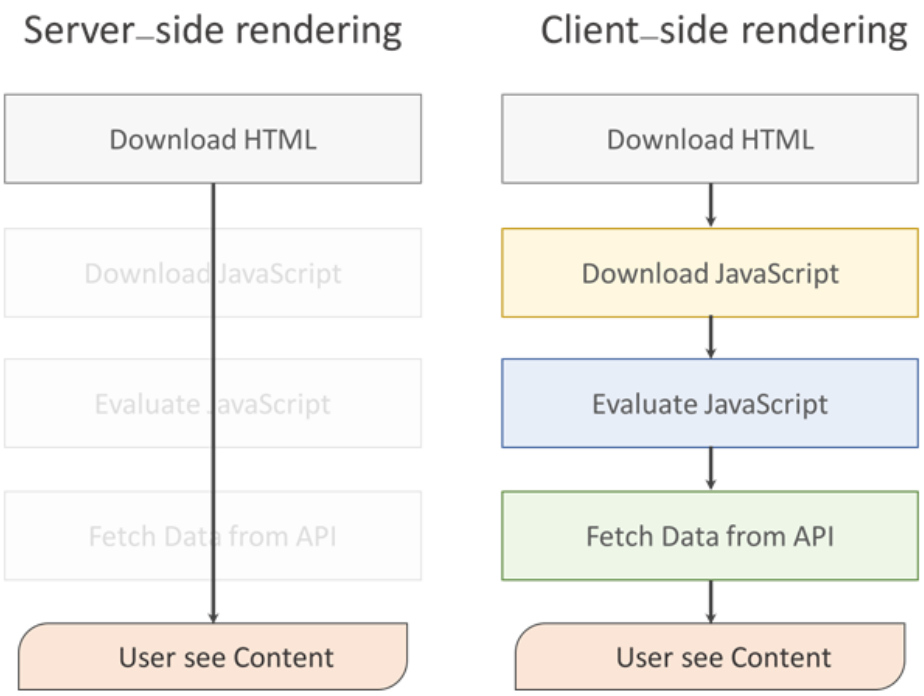
5. 템플릿 엔진

서버 사이드 렌더링

- 템플릿(EJS, JADE)을 사용하여 데이터와 병합한 HTML을 클라이언트로 전송

클라이언트 사이드 렌더링

- 정적인 HTML 템플릿과 동적 데이터 API를 별도로 제공하고 클라이언트 측에서 화면을 구성



5. 템플릿 엔진

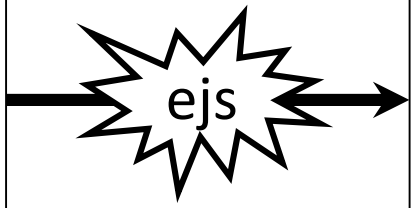
EJS 템플릿 엔진

- 서버 사이드 렌더링을 처리하기 위한 템플릿 엔진
- HTML에 템플릿 코드를 사용하여 가공된 데이터를 출력
- http://ejs.co

EJS 페이지

```
<% var name = "EJS Test"; %>

<h1><%= name %></h1>
<p><%= 52 * 273 %></p>
<hr />
<% for(var i = 0; i < 5; i++) { %>
        <h2>The Square of <%= i %> is <%= i * i%></h2>
<% } %>
```



HTML 페이지

```
<h1>EJS Test</h1>
<p>14196</p>
<hr />
<h2>The Square of 0 is 0</h2>
<h2>The Square of 1 is 1</h2>
<h2>The Square of 2 is 4</h2>
<h2>The Square of 3 is 9</h2>
<h2>The Square of 4 is 16</h2>
```

5. 템플릿 엔진

설치

```
$ npm install ejs --save
```

사용 예

```
app.set('views', './views'); // 템플릿 파일이 위치할 경로
app.set('view engine', 'ejs'); // 사용할 템플릿 엔진 지정

app.get('/', function(request, response){
                response.render('index', { title: 'Hey', message: 'Hello there!'});
});
```

./views/index.ejs

```
<html>
<head>
<title><%= title %></title>
</head>
<body>

<h2><%= message %></h2>

</body>
</html>
```

5. 템플릿 엔진

PUG 템플릿 엔진

- 원래 JADE 템플릿으로 알려져 있었으나 상표권 문제로 PUG로 개명
- 기능면으로 EJS보다 뛰어나지만, HTML에 익숙한 사용자에게는 이질적인 형태라 러닝커브가 있음

설치

```
$ npm install pug --save
```

사용 예

```
app.set('views', './views'); // 템플릿 파일이 위치할 경로
app.set('view engine', 'pug'); // 사용할 템플릿 엔진 지정

app.get('/', function(request, response){
                response.render('index', { title: 'Hey', message: 'Hello there!'});
});
```

./views/index.pug

```
html
  head
    title!= title
  body
    h1!= message
```

6. API 개발 실습

실습 안내

- Todo 리스트 조회, 추가, 삭제 api 만들기
- express, router, body-parser, file system을 사용

스펙

- 텍스트를 저장하고, 특정 텍스트를 삭제할 수 있는 인터페이스를 제공한다.
- 데이터베이스(MySQL, Oracle, MongoDB 등)를 사용하지 않고 파일로 데이터를 관리한다.
- 하나의 todo 는 하나의 파일에 저장한다. (저장 경로: /data/고유키.json)

힌트

고유키 생성	new Date().getTime() 또는 uniqid 모듈 사용
JSON 객체를 문자열로 변환	JSON.stringify()
JSON 문자열을 객체로 파싱	JSON.parse()
디렉토리 내 파일 목록 반환	fs.readdirSync()
파일 생성	fs.writefileSync()
파일 삭세	fs.unlinkSync()

6. API 개발 실습

API 요청, 응답 설계

메서드	URL	요청 데이터	응답 포맷	설명
POST	/todos	{content:"내용"}	JSON	새 todo 추가
GET	/todos		JSON	todo 목록 조회
DELETE	/todos/:id		JSON	특정 todo 삭제

목록 조회 API 응답 데이터 구조

```
{
        "result":"success",
        "data":[
                {"id":1495960066924,"content":"첫번째 할일"},
                {"id":1495960068076,"content":"두번째 할일"},
                {"id":1495960131605,"content":"세번째 할일"},
                {"id":1495960171761,"content":"네번째 할일"},
                {"id":1495960364193,"content":"다섯번째 할일"}
        ]
}
```

6. API 개발 실습

소스 코드

- 저장소
  - oss : https://oss.navercorp.com/changwon-ok/uit-express-todoapp
  - gitlab3 : http://gitlab3.uit.navercorp.com/changwon.ok/uit-express-todoapp
- master 브랜치 : static 파일만 존재(js, css, html)
- feature/express 브랜치 : static + API 개발 완성본 소스

File System 모듈

- 파일 처리와 관련된 내장 모듈

모듈 추출

```
var fs = require('fs');
```

File System 모듈의 메서드

readFile(filepath, encoding, callback)	파일을 읽어서 파일의 내용을 콜백함수로 반환
writeFile(filepath, data, encoding, callback)	파일을 생성
readdir(path, callback)	디렉토리 내 파일 목록을 반환
unlink(filepath, callback)	파일을 삭제

파일 읽기 - readFile

textfile.txt

```
텍스트 샘플
```

readfilejs

```
// 모듈 추출
var fs = require('fs');

// 동기 방식으로 파일 읽기
var text = fs.readFileSync('textfile.txt', 'utf8');
console.log(text);

// 비동기 방식으로 파일 읽기
fs.readFile('textfile.txt', 'utf8', function(error, data){
                console.log(data);
});
```

터미널 실행

```
$ node readfile.js
텍스트 샘플
텍스트 샘플
```

파일 쓰기 - writeFile

writefilejs

```
// 모듈 추출
var fs = require('fs');

// 변수 선언
var data = 'Hello World!!';

// 동기 방식으로 파일 쓰기
fs.writeFileSync('writefileSync.txt', data, 'utf8');
console.log('동기 쓰기 완료');

// 비동기 방식으로 파일 쓰기
fs.writeFile('writefileAsync.txt', data, 'utf8', function(){
                console.log('비동기 쓰기 완료');
});
```

터미널 실행

```
$ node writefile.js
동기 쓰기 완료
비동기 쓰기 완료
```

파일 목록 - readdir

readdir.js

```
// 모듈 추출
var fs = require('fs');

// 동기 방식
var files = fs.readdirSync('./');
console.log(files);

// 비동기 방식
fs.readdir('./', function(error, files){
                    console.log(files);
});
```

터미널 실행

```
$ node readdir.js
[ 'console.js',
  'expressejs',
  'global.js',
  'helloworld.js',
  'helloworld.server.js']
```

파일 삭제 - unlink

```
// 모듈 추출
var fs = require('fs');

// 동기 방식으로 파일 삭제
try{
		fs.unlinkSync('writefileSync.txt');
		console.log('동기식 삭제 성공');
}catch(e){
		console.log('동기식 삭제 실패');
}

// 비동기 방식으로 파일 삭제
fs.unlink('writefileAsync.txt', function(err){
		if(err){
				console.log('비동기식 삭제 실패');
		}else{
				console.log('비동기식 삭제 성공');
		}
});
```

-

End of Document

-

Thank You.

-

TECHNOLOGY SERVICES