

Node.js 기반의 REST API 서버 개발

오리엔테이션

강의 목표

- 노드로 API 서버를 만들 수 있다!
- 테스트 주도 개발을 익힐 수 있다!

강의 대상자

- 자바스크립트로 웹 서비스를 만들어 봤다 🖐️
- 모바일/웹 API 서버를 만들고 싶다 🖐️
- ES6 문법을 안다 🤔

수업 환경 구성

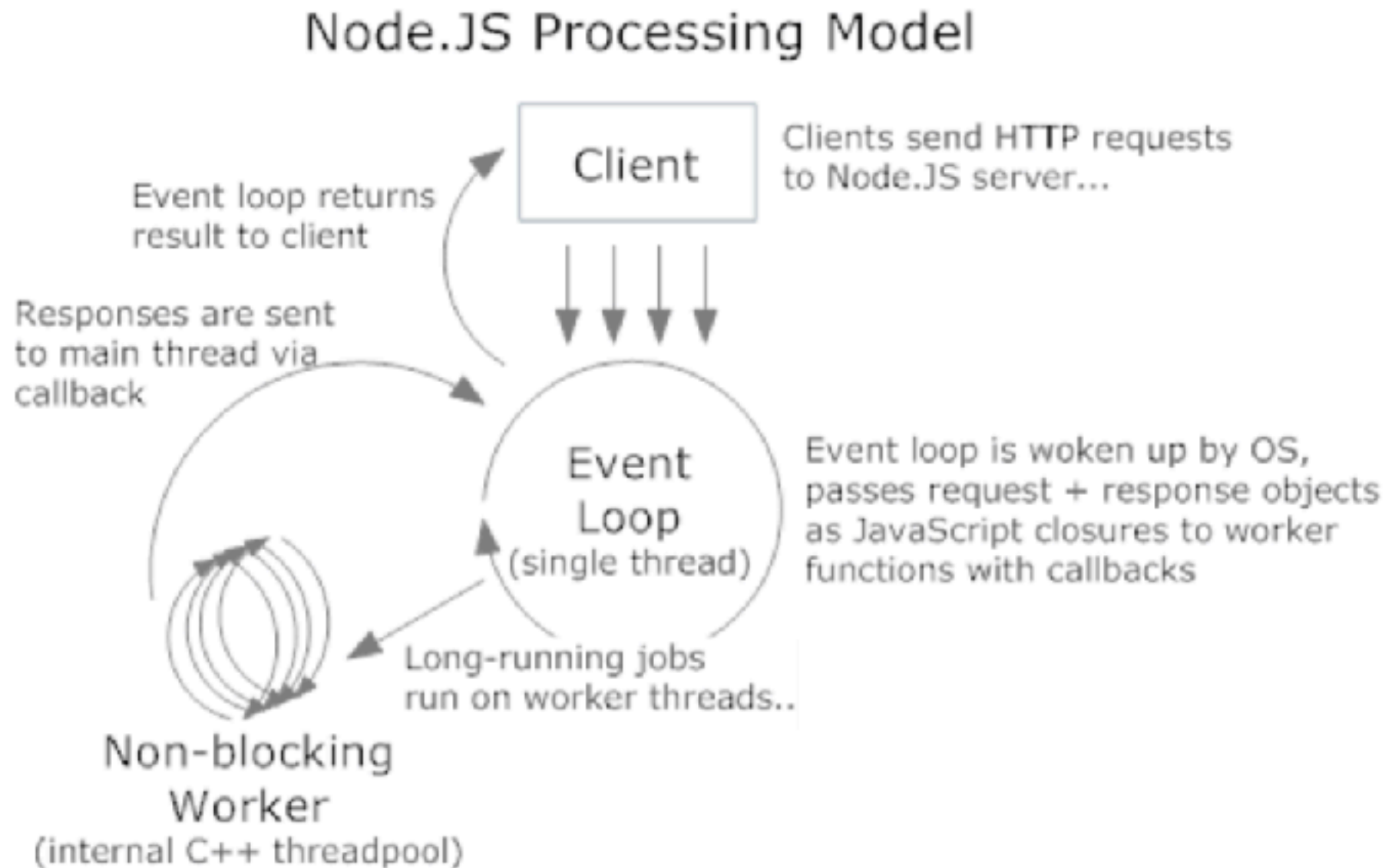
- Node.JS
- 에디터
- 터미널
- Git

Node.JS

노드JS의 기초

- 브라우저 밖에서 자바스크립트 코드를 실행할 수 있다
- 크롬에서 사용하는 V8 엔진을 사용한다 🐼
- 이벤트 기반의 비동기 I/O 프레임워크
- CommonJS를 구현한 모듈 시스템

이벤트 기반의 비동기 I/O 프레임워크



[더 알아보기 >](#)

모듈 시스템

- 브라우저에서는 윈도우 컨텍스트를 사용하거나, RequireJS같은 의존성 로더를 사용함
- 노드는 파일형태로 모듈을 관리할수 있는 CommonJS로 구현
 - 기본 모듈 🐙
 - 써드파티 모듈
 - 사용자 정의 모듈 🐙

기본 모듈

```
const util = require('util')
```

```
const name = 'World'
```

```
const msg = util.format('Hello %s', name)
```

```
console.log(msg) // "Hello World"
```

사용자 모듈

```
// math.js
const math = {
  add(a, b) {
    return a + b
  }
}
module.exports = math
```

```
// index.js
const math = require('./math')
console.log(math.add(1, 2)) // 3
```

비동기 세계

- 노드는 기본적으로 비동기로 동작함
- `readFile()` vs `readFileSync()` 🤖

readFileSync()

```
// test.txt
```

```
테스트 파일입니다.
```

```
// index.js
```

```
const fs = require('fs')
```


```
const file = fs.readFileSync('test.txt', {  
  encoding: 'utf8'  
})
```

```
console.log(file) // "테스트 파일입니다."
```

readFile()

```
const fs = require('fs')  
  
const file = fs.readFile('test.txt', {  
  encoding: 'utf8'  
}, (err, data) => console.log(file)) // "테스트 파일입니다."
```

Hello World 노드 버전

- <https://nodejs.org/dist/latest-v8.x/docs/api/synopsis.html> 

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

```
const hostname = '127.0.0.1';
```

```
const port = 3000;
```

```
const server = http.createServer((req, res) => {  
  res.statusCode = 200;  
  res.setHeader('Content-Type', 'text/plain');  
  res.end('Hello World\n');  
});
```

```
server.listen(port, hostname, () => {  
  console.log(`Server running at http://${hostname}:${port}/`);  
});
```


여기에 새로운 API를 추가하려면?

```
console.log(req.url)
```

```
const server = http.createServer((req, res) => {  
  if (req.url === '/') {  
    res.statusCode = 200;  
    res.setHeader('Content-Type', 'text/plain');  
    res.end('Hello World\n');  
  } else if (req.url === '/users') {  
    res.statusCode = 200;  
    res.setHeader('Content-Type', 'application/json');  
    res.end(JSON.stringify([{name: 'alice'}, {name: 'bek'}]));  
  } else {  
    res.statusCode = 404;  
    res.setHeader('Content-Type', 'text/plain');  
    res.end('Not Found');  
  }  
});
```

Express.JS

익스프레스JS 기초

- 어플리케이션
- 미들웨어
- 라우팅
- 요청/응답 객체

어플리케이션

- 익스프레스 인스턴스를 어플리케이션이라 한다
- 서버에 필요한 기능인 미들웨어를 어플리케이션에 추가한다
- 라우팅 설정을 할 수 있다
- 서버를 요청 대기 상태로 만들수 있다 🐙

미들웨어

- 미들웨어는 함수들의 연속이다
- 로깅 미들웨어를 만들어 보자 🐙
- 써드파티 미들웨어를 사용해 보자 🐙
- 일반 미들웨어 vs 에러 미들웨어

라우팅

- 요청 url에 대해 적절한 핸들러 함수로 연결해 주는 기능을 라우팅이라고 부른다
- 어플리케이션의 `get()`, `post()` 메소드로 구현할 수 있다
- 라우팅을 위한 전용 Router 클래스를 사용할 수도 있다

요청 객체

- 클라이언트 요청 정보를 담은 객체를 요청(Request)객체라고 한다
- http 모듈의 request 객체를 래핑한 것이다
- req.params(), req.query(), req.body() 메소드를 주로 사용한다

응답 객체

- 클라이언트 응답 정보를 담은 객체를 응답(Response)객체라고 한다
- http 모듈의 response 객체를 래핑한 것이다
- res.send(), res.status(), res.json() 메소드를 주로 사용한다

Hello World Express 버전

- <http://expressjs.com/ko/starter/hello-world.html> 

REST API

HTTP 요청

- 모든 자원은 명사로 식별한다
- HTTP 경로로 자원을 요청한다
- 예
 - GET /users
 - GET /users/{id}

HTTP 메서드

- 서버 자원에 대한 행동을 나타낸다. (동사로 표현)
 - GET: 자원을 조회
 - POST: 자원을 생성
 - PUT: 자원을 갱신
 - DELETE: 자원을 삭제
- 이는 익스프레스 어플리케이션의 메소드로 구현되어 있다 🤖

HTTP 상태코드

- 1xx: 아직 처리중
 - 2xx: 자, 여기있어!
 - 3xx: 잘 가~
 - 4xx: 니가 문제임
 - 5xx: 내가 문제임
- 200: 성공(success), GET, PUT
 - 201: 작성됨(created), POST
 - 204: 내용 없음 (No Content), DELETE
 - 400: 잘못된 요청 (Bad Request)
 - 401: 권한 없음 (Unauthorized)
 - 404: 찾을 수 없음 (Not found)
 - 409: 충돌 (Conflict)
 - 500: 서버 에러 (Internal server error)

첫 API 만들기

- GET /users
- 사용자 목록을 조회하는 기능 🐙
- `git checkout -f install-express`

테스트 주도 개발

- TDD로 개발하자!
- mocha, should, superTest

Mocha

- 모카(mocha)는 테스트 코드를 돌려주는 테스트 러너
- 테스트 꾸러미: 테스트 환경으로 모카에서는 describe()으로 구현한다
- 테스트 케이스: 실제 테스트를 말하며 모카에서는 it()으로 구현한다
- `git checkout -f mocha`

Should

- 슈드(should)는 검증(assertion) 라이브러리다
- 가독성 높은 테스트 코드를 만들 수 있다 🐙

SuperTest

- 단위 테스트: 함수의 기능 테스트
- 통합 테스트: API의 기능 테스트
- 슈퍼 테스트는 익스프레스 통합 테스트용 라이브러리다
- 내부적으로 익스프레스 서버를 구동시켜 실제 요청을 보낸 뒤 결과를 검증한다 🐙

TDD로 API 서버 개발

첫 API 테스트 만들기

- 성공
 - 유저 객체를 담은 배열로 응답한다
 - 최대 limit 갯수만큼 응답한다
- 실패
 - limit이 숫자형이 아니면 400을 응답한다
 - offset이 숫자형이 아니면 400을 응답한다

GET /user/:id

- success
 - id가 1인 유저 객체를 반환한다
- error
 - id가 숫자가 아닐경우 400으로 응답한다
 - id로 유저를 찾을수 없을 경우 404로 응답한다

DELETE /users/:id

- success
 - 204를 응답한다
- error
 - id가 숫자가 아닐경우 400으로 응답한다

POST /users

- success
 - 201 상태코드를 반환한다
 - 생성된 유저 객체를 반환한다
 - 입력한 name을 반환한다
- error
 - name 파라미터 누락시 400을 반환한다
 - name이 중복일 경우 409를 반환한다

PUT /users/:id

- success
 - 변경된 name을 응답한다
- error
 - 정수가 아닌 id일 경우 400 응답
 - name이 없을 경우 400 응답
 - 없는 유저일 경우 404 응답
 - 이름이 중복일 경우 409 응답

리팩토링

- 역할에 따라 파일로 분리하자
 - api/user/index.js
 - api/user/user.ctrl.js
 - api/user/user.spec.js

데이터베이스

데이터베이스

- SQL
 - MySQL, PostgreSQL, Aurora, Sqlite
- NoSQL
 - MongoDB, DynamoDB
- In Memory DB
 - Redis, Memcached

SQL 쿼리 기초

- `insert users (`name`) values ('alice');`
- `select * from users;`
- `update users set name = 'bek' where id = 1;`
- `delete from users where id = 1;`

ORM

- 데이터베이스를 객체로 추상화해 논것을 ORM(Object Relational Mapping)이라고 한다
- 쿼리를 직접 작성하는 대신 ORM의 메소드로 데이터 관리할 수 있는 것이 장점이다
- 노드에서 SQL ORM은 시퀀라이저(Sequelize)가 있다

- insert users (` name `) values ('alice');
→ User.create({name: 'alice'})
- select * from users;
→ User.findAll()
- update users set name = 'bek' where id = 1;
→ User.update({name: 'bek'}, {where: {id: 1}});
- delete from users where id = 1;
→ User.destroy({where: {id: 1}});

모델

- 데이터베이스 테이블을 ORM으로 추상화한것을 모델이라고 한다
- 우리가 사용할 유저 모델을 만들어 보자 🐙
 - `sequelize.define()`: 모델 정의
 - `sequelize.sync()`: 데이터베이스 연동

API-디비 연동

- API 로직인 user.ctrl.js에서 모델을 연동하여 디비를 연결한다 🐙

테스트-디비 연동

- 서버 구동과 디비 싱크 로직을 모듈로 분리한다
 - bin/sync-database.js: 데이터베이스 연동
 - bin/www.js: 서버 구동
- 모카 후커로 디비 연동
 - before() / after()
 - beforeEach() / afterEach()

환경의 분리

	Test	Development	Production
Database	sqlite	sqlite	mysql
Logging	None	All	Minimum