

□ 학습 목표

서버사이드에서 자바스크립트를 실행하는 원리 알아보기.  
기술적인 특징인 싱글 스레드, 이벤트루프를 알아보기  
Node.js의 장점과 단점, 테스트 코드 작성.

□ 다루는 내용

Node.js 소개

Node.js의  
실행원리

Node.js의  
기술적인 특징

Node.js 과연  
쓸만한가

Node.js로  
코드작성하기

Node.js  
성능테스트

## 2.1 Node.js 소개

- Node.js는 서버 측 자바스크립트 런타임 환경
- 라이언달이 2009년 오픈소스로 공개
- 브라우저 밖에서 자바스크립트를 사용할 수 있게하는 V8엔진을 사용
- 2010년에는 npm이라는 패키지 매니저 공개
- 2021년 스택오버플로 설문에서는 가장 많이 사용되는 웹 프레임워크에 익스프레스가 3위를 차지할 정도로 최근에는 많이 사용됨.
- I/O에 대한 관점을 완전히 새롭게 해주었다는 점에서 프로그래밍 발전에 중요한 역할을 하였음

## 2.2 Node.js는 서버에서 어떻게 자바스크립트를 실행할까?

### 2.2.1 Node.js의 구성요소

- Node.js의 소스 코드는 C++와 자바스크립트, 파이썬 등으로 이루어져 있음

#### ▼ Node.js의 구성요소

구성요소	설명
Node.js API(자바스크립트)	자바스크립트 API
Node.js 바인딩	자바스크립트에서 C/C++ 함수를 호출할 수 있게 합니다.
Node.js 표준 라이브러리(C++)	운영체제와 관련된 함수들. 타이머(setTimeout), 파일시스템(filesystem), 네트워크 요청(HTTP)
C/C++ 애드온	Node.js에서 C/C++ 소스를 실행할 수 있게 하는 애드온
V8(C++)	오픈 소스 자바스크립트 엔진. 자바스크립트를 파싱, 인터프리터, 컴파일, 최적화에 사용됩니다.
libuv(C++)	비동기 I/O에 초점을 맞춘 멀티플랫폼을 지원하는 라이브러리. 이벤트 루프, 스레드 풀 등을 사용합니다.
기타 C/C++ 컴포넌트	c-ares(DNS), HTTP 파서, OpenSSL, zlib

#### ▼ Node.js의 구성요소

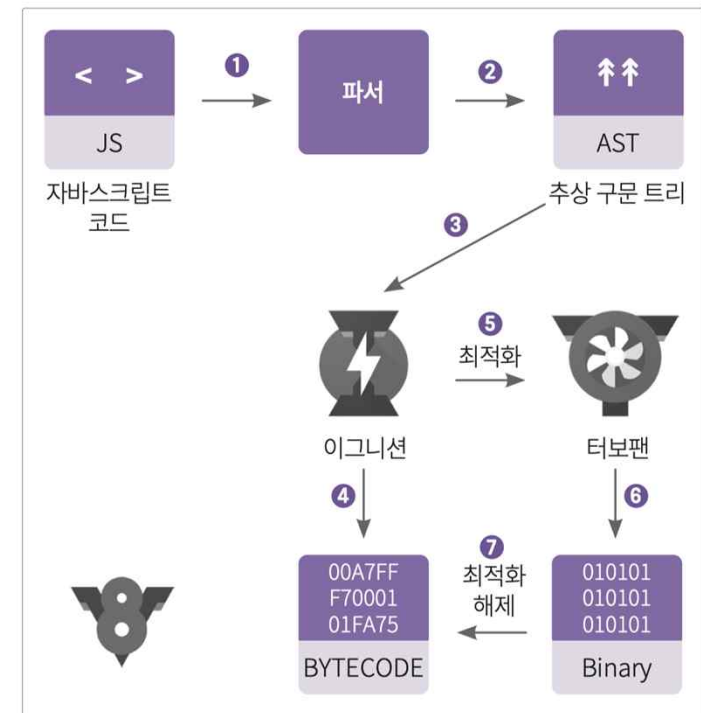


## 2.2 Node.js는 서버에서 어떻게 자바스크립트를 실행할까?

### 2.2.2 자바스크립트 실행을 위한 V8엔진

- V8은 C++로 만든 오픈 소스 자바스크립트 엔진
- 엔진은 파서, 컴파일러, 인터프리터, 가비지 컬렉터, 콜스택, 힙으로 구성

▼ V8 엔진의 자바스크립트 코드 컴파일 단계



## 2.2 Node.js는 서버에서 어떻게 자바스크립트를 실행할까?

### 2.2.3 이벤트 루프와 운영체제 단 비동기 API 및 스레드 풀을 지원하는 libuv

- Node.js에서 HTTP, 파일, 소켓 통신, IO기능등 자바스크립트에 없는 기능은 어떻게 제공하는가? libuv(C++) 라이브러리를 사용

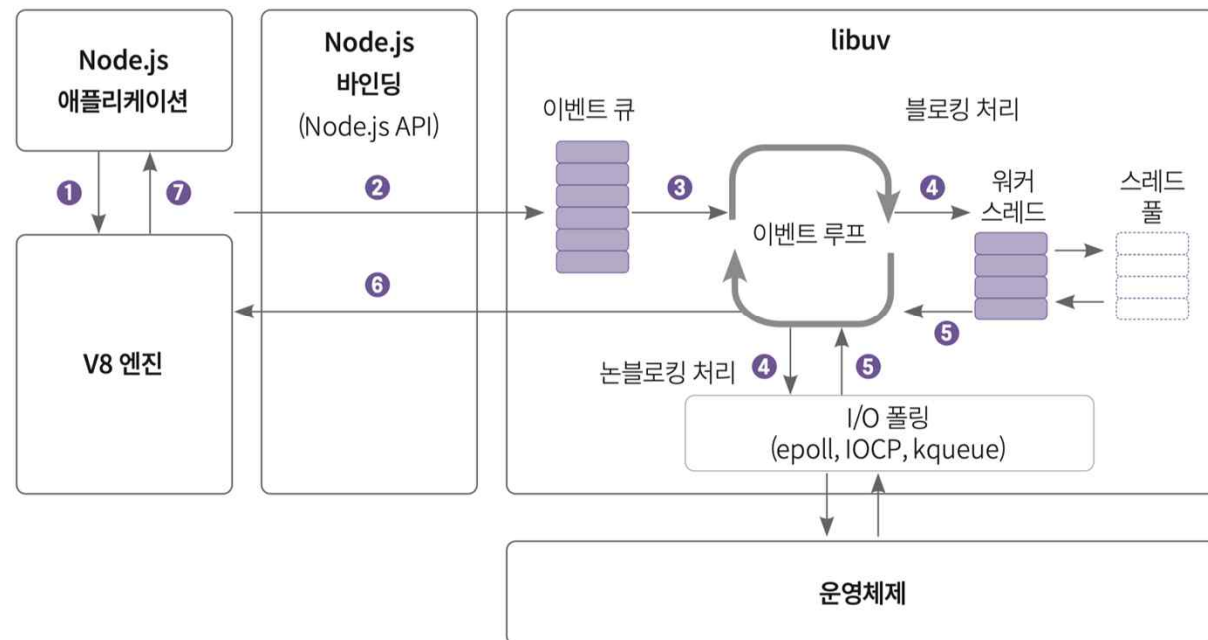
▼ libuv 아키텍처 <sup>10</sup>



## 2.2 Node.js는 서버에서 어떻게 자바스크립트를 실행할까?

### 2.2.4 Node.js 아키텍처

▼ Node.js 아키텍처

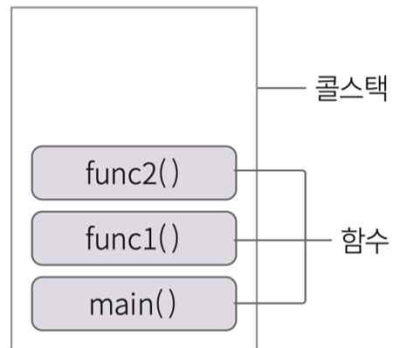


## 2.3 Node.js의 기술적인 특징

### 2.3.1 싱글 스레드

- 싱글 스레드는 콜 스택이 하나만 있다는 것
- 콜 스택 하나라서 한 번에 하나의 작업만 가능

#### ▼ 싱글 스레드의 콜 스택



## 2.3 Node.js의 기술적인 특징

다음 코드의 콜스택 작동 예시

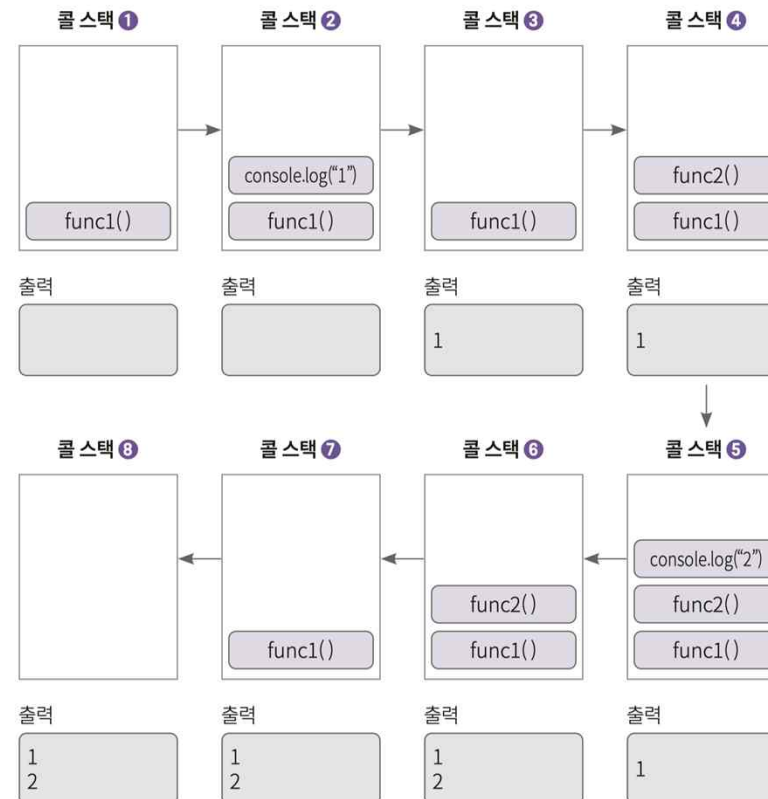
```
function func1() {  
  console.log("1");  
  func2();  
  return;  
}
```

```
function func2() {  
  console.log("2");  
  return;  
}
```

```
func1();
```

```
1  
2
```

▼ 콜스택 작동 예시





## 2.3 Node.js의 기술적인 특징

### 2.3.2 이벤트 기반 아키텍처

- 싱글 스레드는 하나의 요청만 처리 가능

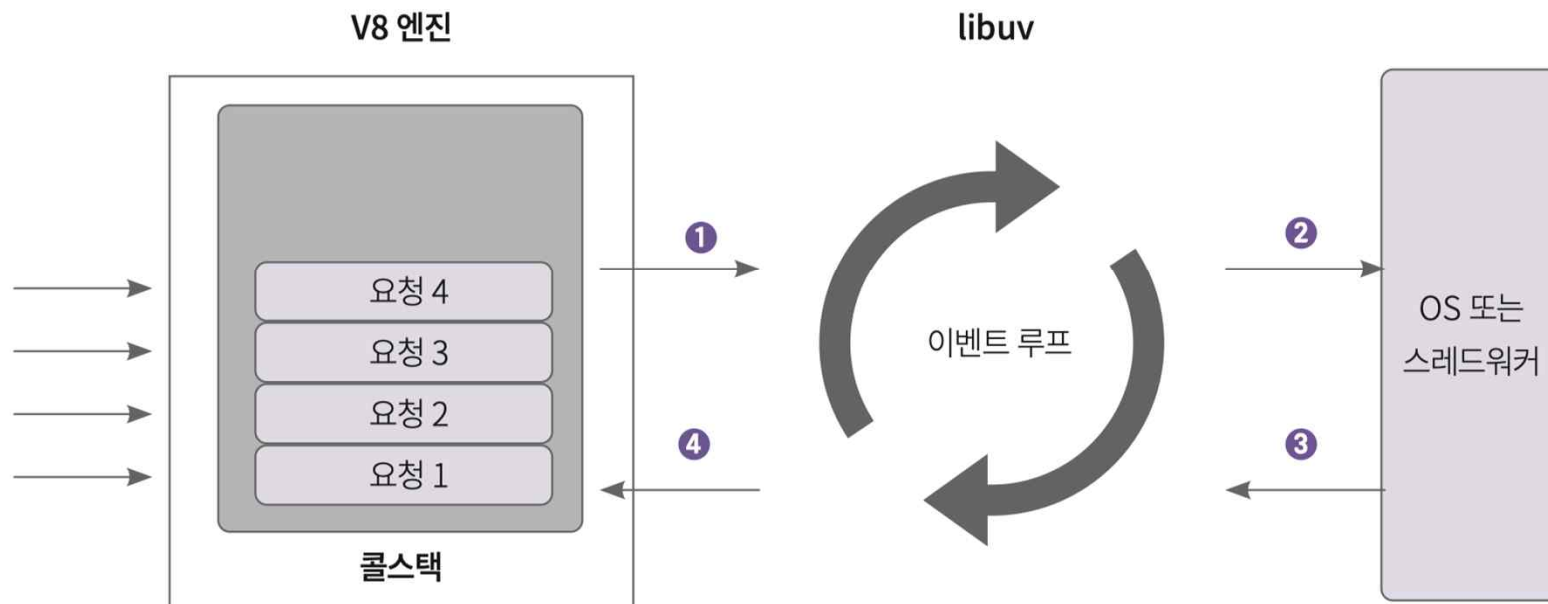
▼ 싱글 스레드에 100개의 요청을 동시에 보냈을 때



- Node.js는 동시처리를 어떻게 할까? > 이벤트 기반 아키텍처 도입

## 2.3 Node.js의 기술적인 특징

### ▼ Node.js의 이벤트 기반 아키텍처

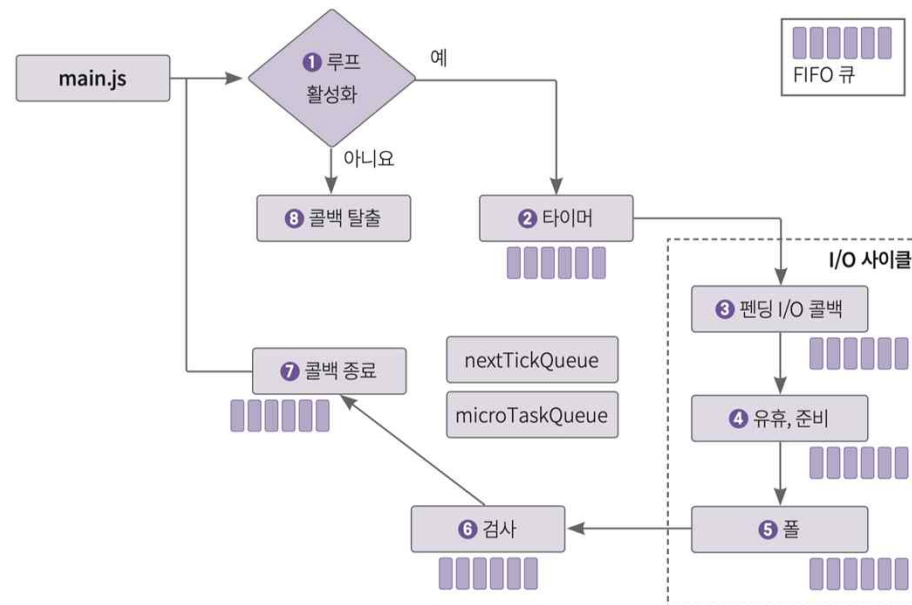


## 2.3 Node.js의 기술적인 특징

### 2.3.3 이벤트 루프

- 이벤트 기반 아키텍처에서 이벤트 루프는 필수이다
- libuv의 이벤트 루프를 사용

▼ Node.js 이벤트 루프의 흐름 <sup>12</sup>



## 2.4 Node.js 과연 쓸 만한가?

### ▼ Node.js의 장점과 단점

장점	단점
<ul style="list-style-type: none"><li>● 비동기 이벤트 기반 IO를 사용해 동시에 여러 요청을 다루기가 용이함</li><li>● 자바스크립트를 사용해 프론트엔드 개발자의 백엔드 진입이 용이함</li><li>● 클라이언트와 같은 언어를 사용하면 서버의 코드에 사용된 로직을 클라이언트에서도 사용할 수 있음</li><li>● 개발자 생태계가 잘 구성되어 있어서, 패키지 매니저에서 필요한 대부분을 제공함</li><li>● V8 엔진이 JIT 컴파일러이므로 서버 기동이 빠름</li></ul>	<ul style="list-style-type: none"><li>● 기본적으로는 CPU를 하나만 사용하므로 멀티코어를 사용하려면 별도의 작업이 필요함</li><li>● 비동기를 지원하지 않는 IO 요청이나 CPU 작업은 주의해서 작업해야 함</li><li>● 콜백을 중첩해서 계속 사용하면 코드 작성 및 디버깅이 힘들어짐</li><li>● 이벤트 기반으로 프로그래밍을 해본 적이 없다면 코드 작성성이 타 언어에 비해 상대적으로 어려울 수 있음</li></ul>

## 2.5 나의 첫 Node.js 서버 프로그램

## 2.6 성능 테스트하기

### 2.6.1 k6 설치

- 윈도우의 경우 msi파일을 받아서 설치
- 맥OS는 brew 를 이용

### 2.6.2 k6로 성능 테스트 스크립트 작성하기

\$ k6.exe run test\_hello.js

```
import http from "k6/http";

export const options = {
  vus: 100,
  duration: "10s",
};

export default function () {
  http.get("http://localhost:8000");
}
```

□ 학습 목표

□ 다루는 내용

Node.js로 “OK”만 반환하는 서버 작성하기.  
라우터 추가 및 리팩터링, 동적으로 응답하도록 변경. 만들었던 것을  
익스프레스로 업그레이드 하여 게시판 API를 작성

OK를 반환하는  
서버 만들기

라우터 만들기

createServer()  
리팩터링 하기

동적으로  
응답하기

라우터  
리팩터링하기

익스프레스  
사용하기

익스프레스로  
API서버만들기

게[시판 API  
테스트하기

## 3.1 OK를 반환하는 서버 만들기

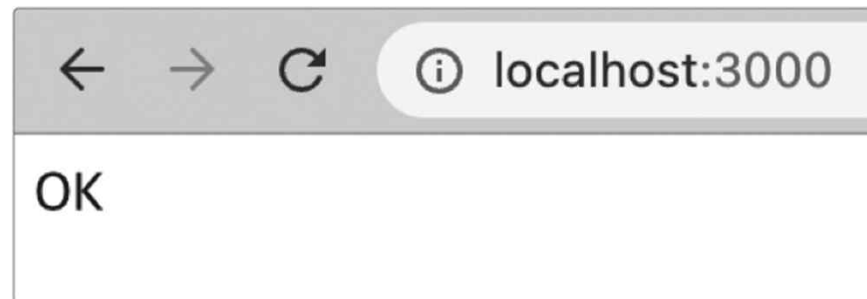
- 모든 요청에 OK를 반환해주는 서버를 구현하자

```
1  const http = require("http");
2  const server = http.createServer((req, res) => {
3    res.setHeader("Content-Type", "text/html");
4    res.end("OK");
5  });
6
7  server.listen("3000", () => console.log("OK서버 시작!"));
```



### 3.1 OK를 반환하는 서버 만들기

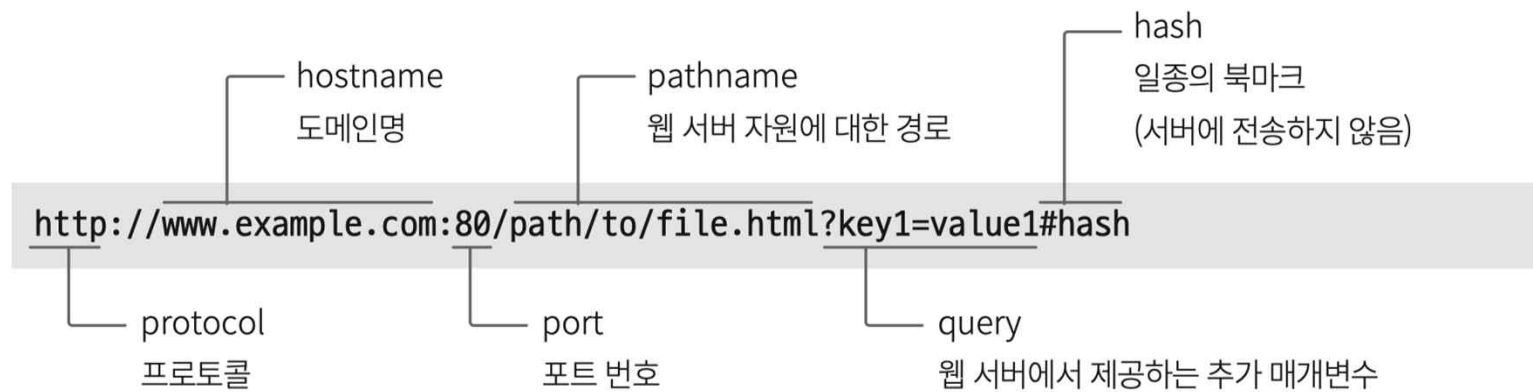
- 모든 요청에 OK를 반환해주는 서버를 구현하자
  - OK가 나오면 성공



## 3.2 라우터 만들기

- URL구조 살펴보기

- ▼ URL의 구조



## 3.2 라우터 만들기

- localhost:3000/user와  
localhost:3000/feed라는  
두 URL에 대한 코드 작성

```
1  const http = require("http");
2  const url = require("url"); // ❶
3  http
4    .createServer((req, res) => {
5      const path = url.parse(req.url, true).pathname; // ❷
6      res.setHeader("Content-Type", "text/html; charset=utf-8");
7
8      if (path === "/user") {
9        res.end("[user] name : andy, age: 30"); // ❸
10     } else if (path === "/feed") {
11       res.end(`<meta charset="UTF-8"><ul>
12         <li>picture1</li>
13         <li>picture2</li>
14         <li>picture3</li>
15       </ul>
16       `); // ❹
17     } else {
18       res.statusCode = 404;
19       res.end("404 page not found"); // ❺
20     }
21   })
22   .listen("3000", () => console.log("라우터를 만들어보자!"));
23
```

## 3.2 라우터 만들기

- localhost:3000/user 접속시



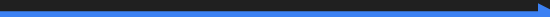
- localhost:3000/feed 접속시

- picture1
- picture2
- picture3

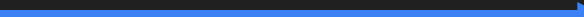
- 그외는 404(페이지없음) 에러 발생

### 3.3 createServer() 리팩터링 하기

```
1  const http = require("http");
2  const url = require("url");
3  http
4    .createServer((req, res) => {
5      const path = url.parse(req.url, true).pathname;
6      res.setHeader("Content-Type", "text/html");
7      if (path === "/user") {
8        user(req, res);
9      } else if (path === "/feed") {
10        feed(req, res);
11      } else {
12        notFound(req, res);
13      }
14    })
15    .listen("3000", () => console.log("라우터를 만들어보자!"));
```



```
const user = (req, res) => {
  res.end(`[user] name : andy, age: 30`);
};
```



```
const feed = (req, res) => {

  res.end(`<ul>
    <li>picture1</li>
    <li>picture2</li>
    <li>picture3</li>
  </ul>
  `);
};
```

## 3.4 동적으로 응답하기

- 3.3까지의 서버는 응답으로 항상 같은 결과값을 보여줌
- user() 함수를 수정하여 매개변수에 따라 응답이 변경되게 수정하자.

```
1 const user = (req, res) => {  
2   const userInfo = url.parse(req.url, true).query;  
3   res.end(`[user] name : ${userInfo.name}, age: ${userInfo.age}`);  
4 };
```

- localhost:3000/user?name=mike&age=20 으로 접속시



## 3.5 라우터 리팩터링하기

- map을 사용하여 라우터를 리팩터링

```
1  http.createServer((req, res) => {
2    const path = url.parse(req.url, true).pathname;
3    res.setHeader("Content-Type", "text/html");
4    if (path in urlMap) {
5      try {
6        urlMap[path](req, res);
7      } catch (err) {
8        console.log(err);
9        serverError(req, res);
10     }
11   } else {
12     notFound(req, res);
13   }
14 })
15 .listen("3000", () => console.log("라우터를 리팩토링해보자!"));
16
17 const urlMap = {
18   "/": (req, res) => res.end("HOME"),
19   "/user": user,
20   "/feed": feed,
21 };
```

## 3.6 익스프레스 프레임워크 사용하기

- 자작 프레임워크로는 한계가 있음. 다음과 같은 기능이 필요

▼ 일반적으로 웹 서버가 제공하는 기능

항목	설명
라우팅	URL 요청을 함수와 매핑시켜주는 기능
정적 파일 서비스	CSS, 자바스크립트, 이미지 등의 정적인 파일을 다루는 기능
템플릿 엔진	동적인 웹페이지를 HTML과 인스턴스를 사용해 생성하는 기능
요청(request) 데이터 다루기	HTTP 요청을 추상화해 편리하게 다룰 수 있게 하는 기능
응답(response) 데이터 다루기	HTTP 응답을 커스터마이징할 수 있는 기능. 파일 내려받기, 이미지 출력 등
파일 업로드	HTTP로 전송된 파일을 읽고 다룰 수 있는 기능
쿠키 및 세션 지원	클라이언트 측 혹은 서버 측의 메모리에 일정 기간 동안 저장해야 하는 데이터를 다루는 기능
리다이렉트	서버의 응답 시 다른 페이지로 전달(redirect)시키는 기능
에러 페이지	요청이 잘못되었거나, 서버 에러 시 특정 에러 페이지를 보여주기
미들웨어	요청 혹은 응답 사이에 공통된 기능을 추가하는 기능



## 3.6 익스프레스 프레임워크 사용하기

- 익스프레스 설치

```
1 npm install express
```

## 3.6 익스프레스 프레임워크 사용하기

- 헬로 Express 만들기

```
1  const express = require("express");
2  const app = express();
3  const port = 3000;
4
5  app.get("/", (req, res) => {
6    res.set({ "Content-Type": "text/html; charset=utf-8" });
7    res.end("헬로 Express");
8  });
9
10 app.listen(port, () => {
11   console.log(`START SERVER : use ${port}`);
12 });
```

## 3.6 익스프레스 프레임워크 사용하기

- 기존 코드를 Express로

```
1  const url = require("url");
2  const express = require("express");
3  const app = express();
4  const port = 3000;
5
6  app.listen(port, () => {
7    console.log("익스프레스로 라우터 리팩토링하기");
8  });
9
10 app.get("/", (_, res) => res.end("HOME"));
11 app.get("/user", user);
12 app.get("/feed", feed);
13
14 function user(req, res) {
15   const user = url.parse(req.url, true).query;
16   res.json(`[user] name : ${user.name}, age: ${user.age}`);
17 }
18
19 function feed(_, res) {
20   res.json(`<ul><li>picture1</li><li>picture2</li><li>picture3</li></ul>`);
21 }
```

## 3.7 익스프레스로 간단한 API 서버 만들기

- 구현할 게시판 API 스펙

### ▼ 게시판 API 스펙

경로	HTTP 메서드	설명
/	get	게시판 목록을 가져옵니다.
/posts	post	게시판에 글을 씁니다. 글은 아이디(id), 제목(title), 작성자(name), 내용(text), 생성일시(createdDt)로 구성됩니다.
/posts/:id	delete	게시글 아이디가 id인 글을 삭제합니다.

## 3.7 익스프레스로 간단한 API 서버 만들기

```
1  const express = require("express");
2  const app = express();
3  let posts = [];
4
5  // req.body를 사용하려면 json 미들웨어를 사용해야한다.
6  // 사용하지 않으면 undefined로 나옴.
7  app.use(express.json());
8
9  // post요청이 application/x-www-form-urlencoded 인 경우 파싱을 위해 사용.
10 app.use(express.urlencoded({ extended: true }));
11
12 app.get("/", (req, res) => {
13   res.json(posts);
14 });
15
16 app.post("/posts", (req, res) => {
17   console.log(typeof req.body);
18   const { title, name, text } = req.body;
19   posts.push(
20     { id: posts.length + 1, title, name, text, createdDt: Date() });
21   res.json({ title, name, text });
22 });
```

다음 슬라이드에 계속...

## 3.7 익스프레스로 간단한 API 서버 만들기

```
1  app.delete("/posts/:id", (req, res) => {
2    const id = req.params.id;
3    const filteredPosts = posts.filter((post) => post.id !== +id);
4    const isLengthChanged = posts.length !== filteredPosts.length;
5    posts = filteredPosts;
6    if (isLengthChanged) {
7      res.json("OK");
8      return;
9    }
10   res.json("NOT CHANGED");
11 });
12
13 app.listen(3000, () => {
14   console.log("welcome board START!");
15 });
16
```

## 3.8 게시판 API 테스트하기

- curl로 테스트

▼ curl 옵션

옵션	사용 예시	설명
-X	-X POST	HTTP 메서드 정보
-d	-d "key1=value1&key2=value2" localhost:3000	POST 통신 시 body 데이터
-H	Content-Type: application/x-www-form-urlencoded	헤더 정보
-x	curl -x http://proxy_server:proxy_port --proxy-user username:password	프록시 서버 설정
-T	curl -T file.txt http://server.com	파일을 서버에 전송 시 사용
-A	curl -A "Mozilla/5.0" http://server.com	유저 에이전트(user agent)를 변경
-i	curl -i https://goldenrabbit.co.kr/	서버의 응답을 결과로 출력
-I	curl -I https://goldenrabbit.co.kr/	서버 응답에서 헤더 값만 출력
-O	curl -O http://server.com/test.txt	서버의 파일을 이름 변경 없이 내려받기
-L	curl -L http://server.com/redirectingURL	리다이렉트 URL 따라가기
-s	curl -s localhost:3000	에러가 발생해도 출력하지 않음(silent)
-S	curl -S localhost:3000	에러 발생 시 에러 출력(Show)

## 3.8 게시판 API 테스트하기

- curl로 GET 호출

```
1 curl -X GET http://localhost:3000
```

- curl로 POST를 호출하여 게시물 등록

```
1 curl -X POST -H "Content-Type: application/x-www-form-urlencoded" \  
2 -d "title=제목1&name=andy&text= 안녕하세요 ~" http://localhost:3000/posts
```

- curl로 DELETE를 호출해 게시물 삭제

```
1 curl -X DELETE localhost:3000/board/2
```