

# NestJS로 배우는 백엔드 프로그래밍 ch 1, 2

#nestjs #typescript

✍ 들어가며

Nestjs 스터디를 위해 작성한 글입니다.

[NestJS로 배우는 백엔드 프로그래밍 | 한용재 - 교보문고 \(kyobobook.co.kr\)](#) 를 교재로 하여, nestjs에 대해 살펴봅니다.

책의 흐름을 따라가되, 필요한 내용을 임의로 레퍼런스하여 작성하였습니다.

가장 하단의 References & Terms에서 레퍼런스를 모아볼 수 있습니다.

## NestJS

### Chapter 1. Hello NestJS

Nest (NestJS)는 효율적이고 확장 가능한 Node.js 서버사이드 애플리케이션을 구축하기 위한 프레임워크입니다. 최신 JavaScript를 사용하며, TypeScript로 작성되었고 이를 완전히 지원합니다. (또한, 순수 JavaScript로도 코딩할 수 있게 해줍니다.)

객체 지향 프로그래밍(OOP), 함수형 프로그래밍(FP), 함수형 반응형 프로그래밍(FRP)의 요소들을 결합한 프레임워크입니다.

FP, FRP 에 대해 알아보려면, [함수형 프로그래밍\(FP\)과 함수형 반응형 프로그래밍\(FRP\)](#)

Nest는 Express(기본값)와 Fastify 같은 강력한 HTTP 서버 프레임워크를 사용합니다.




이러한 일반적인 Node.js 프레임워크들 위에 추상화 계층을 제공하며, 동시에 개발자가 직접 API를 사용할 수 있도록 합니다. 이를 통해 개발자는 기반 플랫폼에 사용할 수 있는 다양한 서드파티 모듈들을 자유롭게 사용할 수 있습니다.

NestJS를 개발하기 위한 정신적 모델은 서버 측 버전의 Angular에게 영감을 받아 작성되었고, 형태는 TypeScript 버전의 Spring Boot와 유사하므로 개발자의 학습 곡선이 낮습니다.

nestjs vs spring boot :

[NestJS vs. Spring Framework In Java - Which Should I Choose? \(selleo.com\)](#)

[NestJS vs Spring Framework | What are the differences? \(stackshare.io\)](#)

 NestJS	 Spring Boot	 Spring Framework																		
<a href="#">+ Follow</a>	<a href="#">+ Follow</a>	<a href="#">+ Follow</a>																		
<a href="#">+ I use this</a>	<a href="#">+ I use this</a>	<a href="#">+ I use this</a>																		
<table><tr><th>Stacks</th><th>Followers</th><th>Votes</th></tr><tr><td>2.4K</td><td>2.9K</td><td>326</td></tr></table>	Stacks	Followers	Votes	2.4K	2.9K	326	<table><tr><th>Stacks</th><th>Followers</th><th>Votes</th></tr><tr><td>25.4K</td><td>23.2K</td><td>1K</td></tr></table>	Stacks	Followers	Votes	25.4K	23.2K	1K	<table><tr><th>Stacks</th><th>Followers</th><th>Votes</th></tr><tr><td>2K</td><td>857</td><td>0</td></tr></table>	Stacks	Followers	Votes	2K	857	0
Stacks	Followers	Votes																		
2.4K	2.9K	326																		
Stacks	Followers	Votes																		
25.4K	23.2K	1K																		
Stacks	Followers	Votes																		
2K	857	0																		

### NestJS의 특징점

NestJS는 Node.js 서버 사이드 애플리케이션을 구축하기 위한 프레임워크로, 다음과 같은 특징점을 가지고 있습니다:

1. **모듈러 아키텍처**: 애플리케이션을 모듈 단위로 구성하여 재사용성과 유지보수성을 높입니다.
2. **타입스크립트 지원**: 강력한 타입 체킹을 통해 코드의 가독성과 안정성을 향상시킵니다.
3. **의존성 주입**: 객체 간의 의존성을 효율적으로 관리하여 테스트 가능성을 높입니다.
4. **데코레이터 사용**: 데코레이터를 통해 코드를 더욱 직관적이고 선언적으로 작성할 수 있습니다.
5. **풍부한 생태계**: Express, Fastify와 같은 다양한 HTTP 프레임워크를 지원하며, GraphQL, WebSockets, Microservices 등 다양한 모듈을 제공합니다.
6. **강력한 테스트 기능**: 유닛 테스트와 통합 테스트를 쉽게 작성할 수 있는 도구와 방법을 제공합니다.

### Express 기반과 Fastify 기반의 차이 및 장단점

#### Express 기반

장점:

- 오랜 역사의 안정성: 많은 커뮤니티 지원과 풍부한 문서 제공.
- 광범위한 미들웨어 지원: 다양한 미들웨어가 존재하여 필요한 기능을 쉽게 추가 가능.

단점:

- 성능 한계: 단일 스레드 이벤트 루프 기반으로, 고성능 요구 상황에서는 성능 저하 가능.

- 유지보수 문제: 오래된 코드베이스로 인해 최신 표준을 완벽히 지원하지 못할 수 있음.

## Fastify 기반

### 장점:

- 고성능: 성능에 최적화된 프레임워크로 높은 처리량과 낮은 지연 시간 제공.
- 비동기/이벤트 기반: 비동기 처리를 기본으로 하여 확장성과 성능을 높임.

### 단점:

- 미들웨어 부족: Express에 비해 미들웨어 생태계가 상대적으로 적음.
- 도입의 복잡성: 기존 Express 기반 코드에서 Fastify로 전환하는 데 추가적인 학습과 코드 변경 필요.

---

nestjs 설치와 생성된 보일러플레이트에 대한 내용은 책으로 설명합니다.

#### Summary

```
npm install -g @nestjs/cli → 전역으로 nestjs CLI 설치
(nvm 사용 권장)
```

```
nest new project-name → 보일러 플레이트 코드 생성 명령어
```

---

## Chapter 2. 웹 개발 기초 지식

### Node.js 알아보기

#### 프론트엔드와 백엔드에서 JavaScript의 차이

JavaScript는 프론트엔드와 백엔드에서 각각 다른 방식으로 사용됩니다.

- **프론트엔드:** **프론트엔드에서는 JavaScript가 웹 브라우저에서 실행** 됩니다. 크로미움 기반 브라우저(예: Google Chrome)는 V8 엔진을 사용하여 JavaScript 코드를 해석하고 실행합니다. 주로 사용자 인터페이스(UI)와 관련된 작업을 수행하며, DOM 조작, 이벤트 처리, 애니메이션, API 호출 등을 담당합니다. 브라우저 환경에서는 보안상의 이유로 파일 시스템 접근, 네트워크 소켓 사용 등의 권한이 제한됩니다.
- **백엔드:** **백엔드에서는 Node.js를 통해 JavaScript가 서버에서 실행** 됩니다. Node.js는 V8 엔진을 사용하여 JavaScript 코드를 실행하지만, 브라우저와 달리 서버 환경에서는 파일 시스템 접근, 네트워크 소켓 사용, 데이터베이스 연동 등 다양한 시스템 자원에 직접 접근할 수 있습니다. 서버 사이드 로직, 데이터베이스 연동, 인증 및 권한 관리, RESTful API 제공 등을 처리하며, 비동기 I/O와 이벤트 루프를 활용하여 높은 동시성을 제공합니다.

---

### Node.js란 무엇인가?

Node.js는 서버 사이드 애플리케이션을 구축하기 위해 만들어진 오픈 소스, 크로스 플랫폼 JavaScript 런타임 환경입니다. 2009년 Ryan Dahl이 처음 개발했으며, JavaScript가 클라이언트뿐만 아니라 서버 측에서도 사용될 수 있도록 설계되었습니다. Node.js는 2008년에 출시된 Google의 V8 JavaScript 엔진을 기반 으로 합니다. V8 엔진은 크로미움 브라우저(예: Google Chrome)에서 사용되는 고성능 JavaScript 엔진으로, 빠른 실행 속도를 제공합니다. Node.js는 비동기 이벤트 기반의 I/O를 지원하여 높은 처리 성능과 확장성을 갖추고 있습니다.

#### Node.js

V8 엔진 기반으로 만든 자바스크립트 서버사이드 런타임

# Node.js

## Node.js Core Library

## Node.js Bindings

V8  
Engine

libuv

@vincent

### npm (Node Package Manager) 설명

npm은 Node.js의 기본 패키지 매니저로, Node.js 애플리케이션에서 사용할 수 있는 다양한 모듈과 패키지를 관리하고 배포하는 도구입니다. 2010년 Isaac Z. Schlueter가 처음 개발했으며, Node.js 생태계의 핵심적인 부분을 차지합니다. npm은 커맨드 라인 인터페이스(CLI)를 통해 패키지를 설치, 업데이트, 제거하는 기능을 제공합니다.

### 단일 스레드에서 구동되는 논블로킹 I/O 이벤트 기반 비동기 방식

Node.js는 단일 스레드에서 구동되는 논블로킹 I/O 이벤트 기반 비동기 방식을 채택하여 높은 처리 성능을 제공합니다. 이 방식은 Node.js가 많은 동시 연결을 효율적으로 처리할 수 있도록 도와줍니다.

### 단일 스레드 모델

Node.js는 기본적으로 단일 스레드에서 동작합니다. 이는 다른 많은 서버 기술들과는 다르게, 하나의 스레드가 모든 요청을 처리한다는 것을 의미합니다. 단일 스레드 모델은 멀티스레드 환경에서 발생할 수 있는 컨텍스트 스위칭 오버헤드를 제거하여 자원 사용을 최적화합니다. 그러나 단일 스레드만으로도 높은 성능을 발휘할 수 있는 이유는 비동기 논블로킹 I/O와 이벤트 루프 덕분입니다.

### 멀티스레딩 방식과의 비교

멀티스레딩 방식에서는 요청마다 새로운 스레드를 생성하거나 스레드 풀에서 스레드를 할당하여 작업을 처리합니다. 이는 여러 작업을 병렬로 처리할 수 있어 장점이 있지만, 몇 가지 단점도 있습니다:

- **메모리 소모:** 스레드가 늘어날수록 메모리 사용량도 증가합니다. 각 스레드는 자체 스택 메모리를 필요로 하므로, 많은 수의 스레드를 생성하면 메모리 사용이 급증할 수 있습니다.
- **공유 자원 관리 문제:** 여러 스레드가 동일한 자원에 접근할 때 동기화 문제가 발생할 수 있습니다. 동기화를 잘못 설정하면 데드락(Deadlock) 상황이 발생할 수 있으며, 이는 시스템이 멈추는 결과를 초래할 수 있습니다.
- **컨텍스트 스위칭 오버헤드:** 스레드 간의 컨텍스트 스위칭은 추가적인 CPU 오버헤드를 발생시킵니다. 이는 시스템 성능을 저하시킬 수 있습니다.

### 싱글스레드와 멀티스레드 비교

특징	싱글스레드 (Node.js)	멀티스레드
동작 방식	하나의 스레드가 모든 작업을 처리	여러 스레드가 병렬로 작업을 처리
장점	<ul style="list-style-type: none"> <li>- 컨텍스트 스위칭 오버헤드 없음</li> <li>- 메모리 사용량 적음</li> <li>- I/O 작업에 유리: 비동기 처리로 많은 동시 연결 처리 가능</li> </ul>	<ul style="list-style-type: none"> <li>- CPU 집약적인 작업에 유리</li> <li>- 병렬 처리로 성능 향상</li> </ul>
단점	<ul style="list-style-type: none"> <li>- CPU 집약적인 작업에 불리</li> <li>- 단일 스레드에서 모든 작업을 처리해야 함</li> </ul>	<ul style="list-style-type: none"> <li>- 메모리 사용량 증가</li> <li>- 동기화 문제 발생 가능</li> <li>- 컨텍스트 스위칭 오버헤드 존재</li> </ul>
적용 분야	<ul style="list-style-type: none"> <li>- I/O 작업이 많은 웹 서버</li> <li>- 높은 동시성을 요구하는 애플리케이션</li> </ul>	<ul style="list-style-type: none"> <li>- CPU 집약적인 작업</li> <li>- 병렬 처리가 필요한 계산 작업</li> </ul>

Node.js가 I/O 작업에 유리한 이유는 비동기 논블로킹 I/O와 이벤트 루프 덕분입니다.

## libuv 와 스레드풀

libuv는 비동기입출력, 이벤트기반에 초점을 두고 OS 커널을 wrapping 한 라이브러리입니다.

Node.js는 단일 스레드 모델을 사용하지만, 내부적으로는 libuv 라이브러리를 통해 스레드풀을 관리합니다. libuv는 Node.js의 비동기 I/O를 지원하는 라이브러리로, 네트워크 작업, 파일 시스템 작업 등의 비동기 작업을 처리합니다. 일부 작업은 스레드풀에서 처리되며, 이는 다음과 같은 장점을 제공합니다:

- **백그라운드 처리:** 파일 시스템 작업이나 DNS 조회와 같은 블로킹 작업을 백그라운드에서 처리하여 메인 스레드의 블로킹을 방지합니다.
- **다중 작업 처리:** 여러 개의 I/O 작업을 병렬로 처리하여 성능을 향상시킵니다.

## CPU 코어 분산 관리

Node.js는 기본적으로 단일 스레드에서 실행되지만, 실제 웹서버 운영 시에는 여러 개의 CPU 코어를 활용할 수 있습니다. cluster 모듈을 사용하면 여러 개의 Node.js 프로세스를 생성하여 각 프로세스가 별도의 코어에서 실행되도록 할 수 있습니다. 이를 통해 멀티코어 시스템의 성능을 극대화할 수 있습니다.

```
const cluster = require('cluster');
const http = require('http');
const numCPUs = require('os').cpus().length;

if (cluster.isMaster) {
  console.log(`Master ${process.pid} is running`);

  // 워커 프로세스를 포크합니다.
  for (let i = 0; i < numCPUs; i++) {
    cluster.fork();
  }

  cluster.on('exit', (worker, code, signal) => {
    console.log(`worker ${worker.process.pid} died`);
  });
} else {
  // 워커 프로세스는 HTTP 서버를 생성합니다.
  http.createServer((req, res) => {
    res.writeHead(200);
    res.end('hello world\n');
  }).listen(8000);

  console.log(`Worker ${process.pid} started`);
}
```

## 논블로킹 I/O

논블로킹 I/O는 입출력 작업이 완료될 때까지 대기하지 않고, 다음 작업을 계속 진행할 수 있게 합니다. 예를 들어, 파일을 읽거나 네트워크 요청을 보낼 때, 해당 작업이 완료될 때까지 기다리지 않고 다른 작업을 처리합니다. 이로 인해 Node.js는 많은 수의 I/O 요청을 효율적으로 처리할 수 있습니다. 논블로킹 I/O 호출은 즉시 반환되어 다른 작업을 수행할 수 있게 합니다. 작업이 완료되면 콜백 함수나 다른 비동기 메커니즘을 통해 결과가 처리됩니다.

## 비동기 I/O

비동기 I/O는 작업이 시작되면 그 작업이 완료될 때까지 기다리지 않고, 다른 작업을 계속 진행하는 방식입니다. 작업이 완료되면 콜백 함수나 프로미스(Promise) 등을 통해 결과가 처리됩니다. Node.js에서 비동기 I/O는 주로 논블로킹 I/O와 함께 사용되며, 이를 통해 고성능 비동기 처리를 구현할 수 있습니다.

### ㉠ 블로킹? 논블로킹? 동기? 비동기?

**블로킹/논블로킹:** 함수가 제어권(함수 실행권)을 언제 넘겨주느냐의 차이입니다.

- **블로킹:** 호출된 함수가 작업을 완료할 때까지 제어권을 유지 하고, 작업이 완료되면 제어권을 호출한 쪽으로 넘겨줍니다. 즉, 작업이 끝날 때까지 다른 작업을 진행할 수 없습니다.

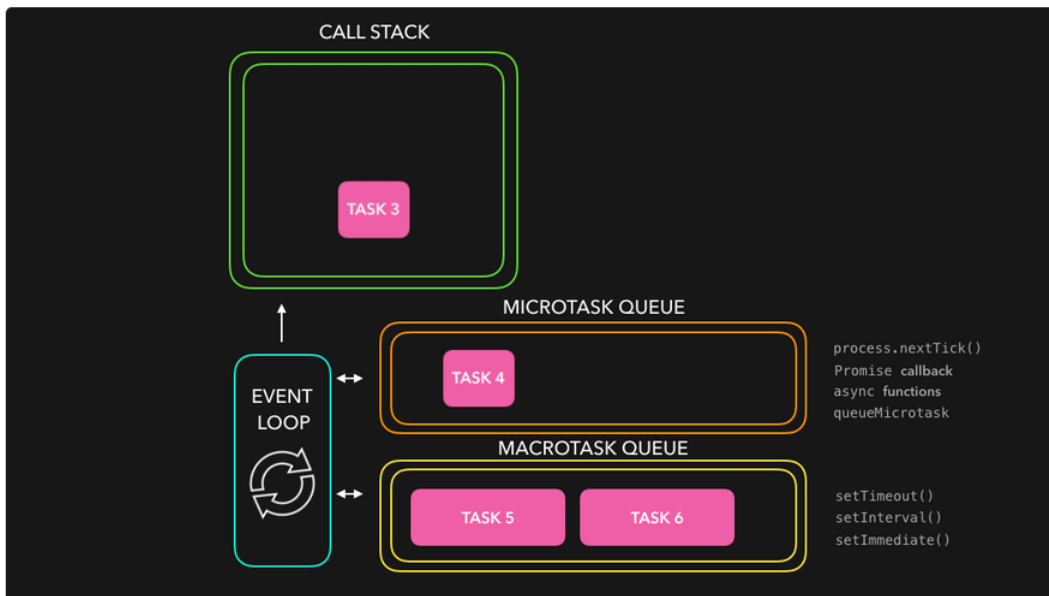
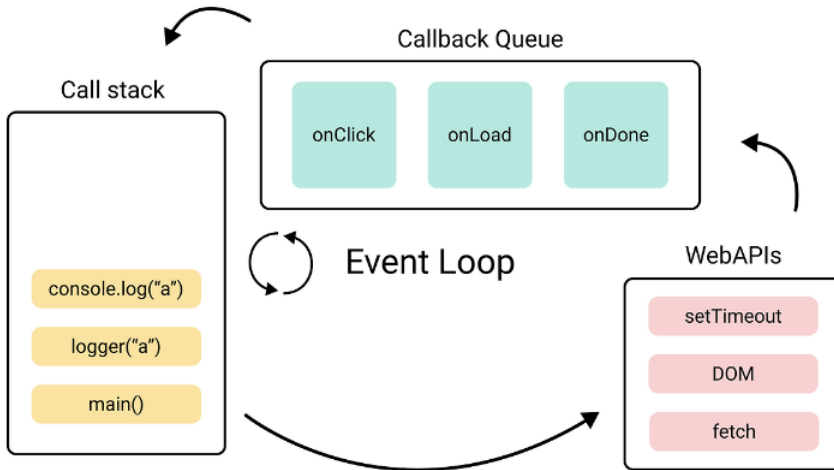
- **논블로킹**: 호출된 함수가 작업을 바로 시작하지만, 제어권을 즉시 호출한 쪽으로 넘겨줍니다. 작업이 완료되지 않아도 다른 작업을 계속 진행할 수 있습니다.

**동기/비동기**: 함수가 작업을 완료했는지를 누가 체크하느냐의 차이입니다.

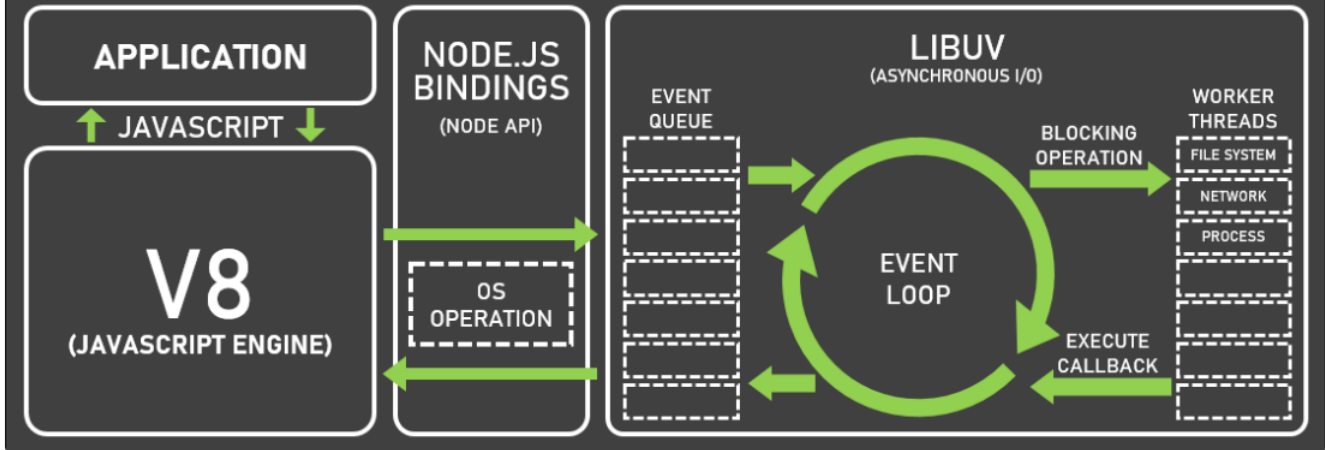
- **동기**: 호출된 함수가 작업이 완료되면 결과를 반환합니다. 호출한 측은 결과가 반환될 때까지 대기 합니다.
- **비동기**: 호출된 함수가 작업을 시작하고, 완료 여부를 함수 자신이 아닌, 나중에 별도의 방식(콜백 함수, 프로미스 등)으로 알립니다. 호출한 측은 함수의 완료를 기다리지 않고 다른 작업을 계속 할 수 있습니다.

## JavaScript 이벤트 루프

자바스크립트는 단일 스레드 언어로, 동시에 하나의 작업만을 실행할 수 있습니다. 하지만 실제로는 비동기적으로 여러 작업을 처리할 수 있는 것처럼 보입니다. 이러한 비동기 작업 처리의 비밀은 바로 이벤트 루프에 있습니다. 특히 Node.js 환경에서는 libuv 라이브러리가 이 이벤트 루프를 관리합니다.



# THE NODE.JS SYSTEM



Event Queue : 비동기 I/O 작업 결과를 저장하고 처리하기 위한 자료구조 (웹브라우저의 Task Queue와 비슷하다)

## 1. 호출 스택 (Call Stack)

호출 스택은 현재 실행 중인 함수의 실행 컨텍스트를 저장하는 자료구조입니다. 자바스크립트는 스택 기반 언어로, 함수가 호출되면 그 함수의 실행 컨텍스트가 스택에 푸시되고, 함수 실행이 완료되면 스택에서 팝됩니다.

```
function foo() {  
  console.log('foo');  
}  
  
function bar() {  
  foo();  
  console.log('bar');  
}  
  
bar();
```

위 코드가 실행될 때, 호출 스택의 상태는 다음과 같습니다:

1. bar 함수가 호출되어 호출 스택에 푸시됩니다.
2. bar 내부에서 foo 함수가 호출되어 호출 스택에 푸시됩니다.
3. foo 함수가 실행을 마치고 호출 스택에서 팝됩니다.
4. bar 함수가 실행을 마치고 호출 스택에서 팝됩니다.

## 2. Web APIs / Node APIs

비동기 작업(예: 타이머, 네트워크 요청, 이벤트 리스너 등)은 호출 스택에서 브라우저의 경우 Web APIs, Node.js의 경우 Node APIs로 이동합니다. 이 API들은 브라우저나 Node.js에서 제공하는 백그라운드 작업을 처리합니다. Node.js에서는 주로 libuv 라이브러리를 통해 이러한 작업이 관리됩니다.

예를 들어, 다음과 같은 코드가 있을 때:

```
import { readFile } from 'fs';  
  
console.log('Start');  
  
readFile('example.txt', 'utf8', (err, data) => {  
  if (err) {  
    console.error(err);  
    return;  
  }  
  console.log(data);  
});  
  
console.log('End');
```

이 코드는 다음과 같은 순서로 실행됩니다:

1. console.log('Start') 가 호출 스택에 푸시되고, 실행된 후 팝됩니다.
2. readFile 함수가 호출 스택에 푸시되고, 파일 읽기 작업이 libuv를 통해 비동기로 처리됩니다.

3. `readFile` 이 호출 스택에서 팝됩니다.
4. `console.log('End')` 가 호출 스택에 푸시되고, 실행된 후 팝됩니다.
5. 파일 읽기 작업이 완료되면, 콜백 함수가 콜백 큐에 추가됩니다.

### 3. 콜백 큐 (Callback Queue)

콜백 큐는 비동기 작업이 완료된 후 콜백 함수를 대기시키는 큐입니다. 호출 스택이 비어 있을 때 이벤트 루프는 콜백 큐에서 콜백 함수를 꺼내어 호출 스택에 푸시합니다.

### 4. 이벤트 루프 (Event Loop)

이벤트 루프는 호출 스택이 비어 있을 때 콜백 큐에 있는 콜백 함수를 호출 스택으로 옮깁니다. 이 과정은 반복되며, 이를 통해 비동기 콜백 함수가 실행됩니다. 이벤트 루프는 여러 단계로 나뉘어 있으며, 각 단계마다 특정한 종류의 작업을 처리합니다.

각 단계는 시스템 실행 한도에 다다르게 되면, 콜백이 큐에 있어도 다음 단계로 넘어가게 됩니다.

#### 이벤트 루프(매크로 태스크)의 단계

브라우저에서 매크로 태스크로 불리는 단계는 node.js에서 더 세분화되어서 이벤트 큐로서 관리된다.



1. **Timers:** `setTimeout` 과 `setInterval` 같은 타이머로 만들어진 콜백이 이 단계에서 실행됩니다. `timer_queue` 에서 꺼내서 실행합니다.
2. **Pending Callbacks:** 지연된 I/O 콜백이 이 단계에서 실행됩니다. `pending_queue` 에서 대기중인 콜백이 있다면 꺼내어 실행합니다.
3. **Idle, Prepare:** 내부적으로 Node.js에서 사용됩니다. (틱마다 실행됩니다. 각 단계를 이동하는 것을 틱이라고 합니다.)
4. **Poll:** 새로운 I/O 이벤트를 가져오고 I/O 관련 콜백을 실행합니다. `watch_queue` 에서 콜백을 가져와서 실행합니다. (파일 시스템의 변경을 감지하는 큐) 모든 `watch_queue` 를 실행한 경우에는 `check_queue` , `pending_queue` 를 확인하고 실행합니다.
5. **Check:** `setImmediate` 콜백이 이 단계에서 실행됩니다.
6. **Close Callbacks:** 닫기 콜백이 이 단계에서 실행됩니다. 이때 소켓의 `close` 와 같은 `destroy` 타입의 이벤트가 실행됩니다.

### 7. 넥스트틱 큐 (Next Tick Queue)

- 설명: `process.nextTick` 콜백이 이 큐에 들어갑니다.
- 처리 순서: 매크로태스크 큐와 마이크로태스크 큐보다 높은 우선 순위를 가집니다. 매크로태스크가 끝난 후, 마이크로태스크 큐보다 먼저 처리됩니다.

### 8. 마이크로태스크 큐 (Micro Task Queue)

- 설명: `Promise` 의 `then` , `catch` , `finally` 콜백이 이 큐에 들어갑니다.
- 처리 순서: 넥스트틱 큐 다음에 처리됩니다. 각 매크로태스크가 끝난 후, 넥스트틱 큐와 마이크로태스크 큐가 처리됩니다.

#### 정리) Node.js의 이벤트 루프 순서

1. 호출 스택(Call Stack): 현재 실행 중인 동기 작업이 저장됩니다.
2. 넥스트틱 큐(Next Tick Queue): `process.nextTick` 콜백이 처리됩니다.
3. 마이크로태스크 큐(Micro Task Queue): `Promise` 의 `then` , `catch` , `finally` 콜백이 처리됩니다.
4. 타이머 단계(Timers Phase): `setTimeout` 과 `setInterval` 콜백이 처리됩니다.
5. 펜딩 콜백 단계(Pending Callbacks Phase): 지연된 I/O 콜백이 처리됩니다.
6. 아이들, 준비 단계(Idle, Prepare Phase): 내부적으로 사용됩니다.
7. 폴 단계(Poll Phase): 새로운 I/O 이벤트를 가져오고, I/O 관련 콜백을 실행합니다.
8. 체크 단계(Check Phase): `setImmediate` 콜백이 처리됩니다.
9. 클로즈 콜백 단계(Close Callbacks Phase): 닫기 콜백이 처리됩니다.

## 패키지 의존성 관리

책 내용으로 대체

## 타입스크립트

타입스크립트는 마이크로소프트에서 개발한 오픈 소스 프로그래밍 언어로, 자바스크립트의 상위 집합(superset)입니다. 타입스크립트는 자바스크립트의 모든 기능을 포함하면서도 정적 타입 검사와 최신 자바스크립트 기능을 추가적으로 제공합니다. 이를 통해 개발자들이 더 안정적이고 유지 보수하기 쉬운 코드를 작성할 수 있게 합니다.

### 타입스크립트의 특징

#### 1. 정적 타입 검사 (Static Type Checking):

타입스크립트는 변수, 함수, 객체 등의 타입을 명시적으로 정의할 수 있습니다. 이를 통해 컴파일 단계에서 타입 오류를 잡을 수 있습니다.

```
let message: string = "Hello, TypeScript";
// message = 42; // 오류: 'number' 형식은 'string' 형식에 할당할 수 없습니다.
```

#### 2. 최신 자바스크립트 기능 지원:

타입스크립트는 ES6+ (ECMAScript 2015 이상의 버전) 기능을 지원합니다. 예를 들어, 화살표 함수, 클래스, 비구조화 할당 등의 최신 기능을 사용할 수 있습니다.

#### 3. 객체 지향 프로그래밍 지원:

타입스크립트는 클래스, 인터페이스, 상속 등 객체 지향 프로그래밍(OOP)의 개념을 지원합니다.

### 트랜스파일 언어의 특징

타입스크립트는 트랜스파일(transpile) 언어로, 작성된 타입스크립트 코드를 자바스크립트 코드로 변환(트랜스파일)하여 실행할 수 있게 합니다.

- 트랜스파일:** 소스 코드를 다른 프로그래밍 언어의 소스 코드로 변환하는 과정입니다.
- 타입스크립트 코드는 실행되기 전에 자바스크립트 코드로 변환되며, 이 자바스크립트 코드는 모든 브라우저와 Node.js 환경에서 실행될 수 있습니다.

#### 1. 타입스크립트 설치

타입스크립트를 설치하려면, npm(Node Package Manager)을 사용합니다.

```
npm install -g typescript
```

#### 2. 타입스크립트 파일 작성

- 예시: example.ts

```
const greeting: string = "Hello, TypeScript!";

console.log(greeting);`
```

#### 3. 타입스크립트 파일을 자바스크립트로 트랜스파일

- tsc 명령어를 사용하여 타입스크립트 파일을 자바스크립트 파일로 변환합니다.  
tsc example.ts
- 이 명령어를 실행하면 example.js 파일이 생성됩니다. 이 파일은 자바스크립트로 변환된 코드입니다.

```
var greeting = "Hello, TypeScript!";

console.log(greeting);
```

### 타입스크립트의 타입

**원시값(Primitive Values):** 변하지 않는 값으로, 직접 해당 값을 저장합니다.

타입	자바스크립트(JavaScript)	타입스크립트(TypeScript)
string	지원	지원
number	지원	지원
boolean	지원	지원
null	지원	지원
undefined	지원	지원
symbol	지원 (ES6부터)	지원
bigint	지원 (ES2020부터)	지원

- string: 문자열 데이터 타입



- `number`: 숫자 데이터 타입 (정수와 부동 소수점 모두 포함)
- `boolean`: 논리 데이터 타입 ( `true` 또는 `false` )
- `null`: 아무런 값도 없음을 나타내는 데이터 타입
- `undefined`: 값이 정의되지 않음을 나타내는 데이터 타입
- `symbol`: 고유하고 변경 불가능한 데이터 타입 (ES6부터 지원)
- `bigint`: 매우 큰 정수를 표현하는 데이터 타입 (ES2020부터 지원)

**참조값(Reference Values):** 객체를 참조하는 값으로, 메모리 주소를 저장합니다.

타입	자바스크립트(JavaScript)	타입스크립트(TypeScript)
<code>object</code>	지원	지원
<code>array</code>	지원	지원
<code>function</code>	지원	지원
<code>class</code>	지원 (ES6부터)	지원
<code>interface</code>	지원하지 않음	지원
<code>tuple</code>	지원하지 않음	지원
<code>enum</code>	지원하지 않음	지원
<code>type alias</code>	지원하지 않음	지원

- `object`: 객체 데이터 타입
- `array`: 배열 데이터 타입
- `function`: 함수 데이터 타입
- `class`: 클래스 데이터 타입 (ES6부터 지원)
- `interface`: 타입스크립트에서 사용되는 인터페이스
- `tuple`: 타입스크립트에서 사용되는 튜플
- `enum`: 타입스크립트에서 사용되는 열거형
- `type alias`: 타입스크립트에서 사용되는 타입 별칭

**기타 타입 (TypeScript 전용)**

타입	자바스크립트(JavaScript)	타입스크립트(TypeScript)
<code>any</code>	지원하지 않음	지원
<code>unknown</code>	지원하지 않음	지원
<code>void</code>	지원하지 않음	지원
<code>never</code>	지원하지 않음	지원

- `any`: 모든 타입을 허용하는 타입 (타입 검사를 무시)
- `unknown`: 알 수 없는 타입 (런타임까지 타입을 확인할 수 없음)
- `void`: 반환 값이 없음을 나타내는 타입 (주로 함수의 반환 타입으로 사용)  
-> 명시적으로 반환값이 없는 상황
- `never`: 절대 발생하지 않는 값을 나타내는 타입 (함수가 결코 반환하지 않음을 나타냄)  
-> 항상 `throw new Error();` , 무한 루프 같은 상황

[기본 타입 | 타입스크립트 핸드북 \(joshua1988.github.io\)](https://joshua1988.github.io)  
[TypeScript: Documentation - Do's and Don'ts \(typescriptlang.org\)](https://www.typescriptlang.org/)

**References & Terms**

[Documentation | NestJS - A progressive Node.js framework](#)  
[NestJS | Technology Radar | Thoughtworks](#)  
[NestJS vs. Spring Framework In Java - Which Should I Choose? \(selleo.com\)](#)  
[NestJS vs Spring Framework | What are the differences? \(stackshare.io\)](#)  
  
[Express vs Fastify Performance. Whose performance is better? | by Kishan Patel | SixBerries Labs | Medium](#)  
[I made Express faster than Fastify \(100x faster JSON, also NestJS\) - DEV Community](#)  
  
[Node.js - 위키백과, 우리 모두의 백과사전 \(wikipedia.org\)](#)  
[Node.js — Run JavaScript Everywhere \(nodejs.org\)](#)  
[npm Docs \(npmjs.com\)](#)  
  
[GitHub - libuv/libuv: Cross-platform asynchronous I/O](#)  
[블로킹\(Blocking\)/논블로킹\(Non-Blocking\), 동기\(Sync\)/비동기\(Async\) 구분하기 \(tistory.com\)](#)  
[Node.js — The Node.js Event Loop \(nodejs.org\)](#)

[자바스크립트 이벤트 루프 동작 구조 & 원리 끝판왕 \(tistory.com\)](http://tistory.com).

[libuv의 이벤트루프 맛보기 하루 1mm 개발 노트 \(tistory.com\)](http://tistory.com).

[Node.js 이벤트 루프\(Event Loop\) 살살이 분석하기 | 쿠키의 개발 블로그 \(korecmblog.com\)](http://korecmblog.com)[Visualizing The Timer Queue in Node.js Event Loop \(builder.io\)](http://builder.io).

[기본 타입 | 타입스크립트 핸드북 \(joshua1988.github.io\)](http://joshua1988.github.io).

[소개 · GitBook \(typescript-kr.github.io\)](http://typescript-kr.github.io).

[TypeScript: Documentation - Do's and Don'ts \(typescriptlang.org\)](http://typescriptlang.org).

---

## Appendix

1. [함수형 프로그래밍\(FP\)](#)과 [함수형 반응형 프로그래밍\(FRP\)](#).

---

[첫 걸음 - 쉽게 풀어 쓴 Nest.js \(wisewiredbooks.com\)](http://wisewiredbooks.com).

[기본 타입 · GitBook \(typescript-kr.github.io\)](http://typescript-kr.github.io).