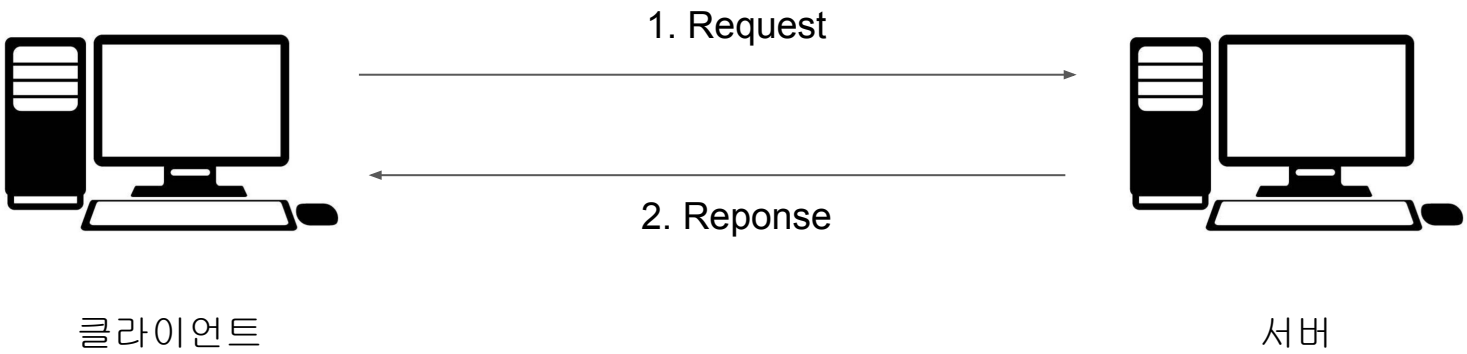
1장. 노드 시작하기

1.1 핵심 개념 이해하기

Node.js는 Chrome V8 Engine으로 빌드된 Javascript 런타임

1.1.1 서버



1.1.2 자바스크립트 런타임

Node.js = Node.js Core Library + Node.js Bindings + Chrome V8 + libuv

1.1.2 Chrome V8 Engine란

What is ‘Chrome V8 Engine’?

=> 구글이 주도하여 C++로 작성된 **고성능의 자바스크립트 & 웹 어셈블리 엔진**

Why made it?

=> 기존의 인터프리터가 **너무 느림**

Who uses it?

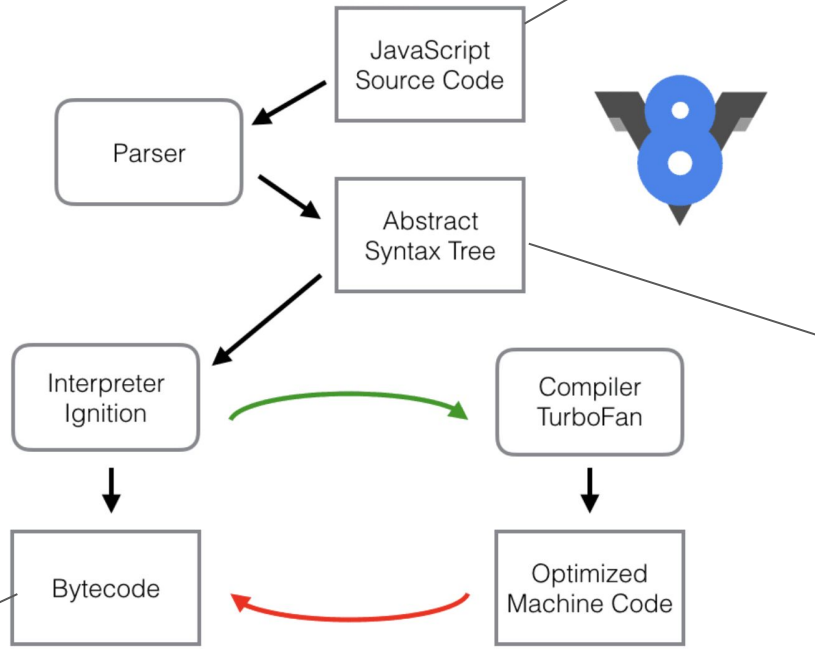
=> Chrome, Node.js,

1.1.2 Chrome V8 Engine 특징

- C++
- ECMAScript
- JIT Compiler

1.1.2 Chrome V8 Engine

```
function hello (name) {
  return 'Hello,' + name;
}
```




```
{
  type: 'FunctionDeclaration',
  name: 'hello'
  arguments: [
    {
      type: 'Variable',
      name: 'name'
    }
  ]
  // ...
}
```

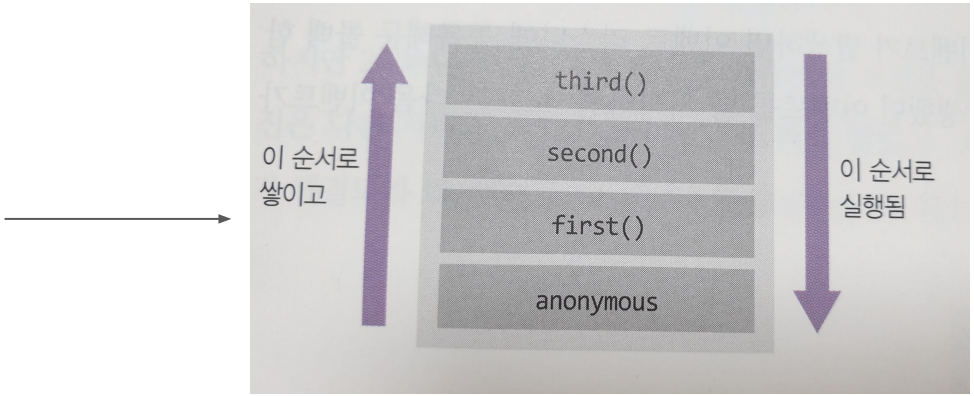
```
a0                  StackCheck
12 00               LdaConstant [0]
26 fb               Star r0
25 02               Ldar a0
32 fb 00            Add r0, [0]
a4                  Return
```

1.1.3 이벤트 기반

- 이벤트 리스너에 콜백 함수를 등록
- 예) 파일 읽기 작업을 마치면 -> afterRead()함수를 실행
- 예) 네트워크 요청이 오면 -> writeLog()함수를 실행

1.1.3 이벤트 기반

```
function first() {
  second();
  console.log('첫 번째');
}
function second() {
  third();
  console.log('두 번째');
}
function third() {
  console.log('세 번째');
}
first();
```



1.1.3 이벤트 기반

```
function run() {
  console.log('3초 후 실행');
}
console.log('시작');
setTimeout(run, 3000);
console.log('끝');
```

시작
끝
3초 후 실행

이것은 이벤트 루프로
가능한 것

1.1.3 이벤트 루프

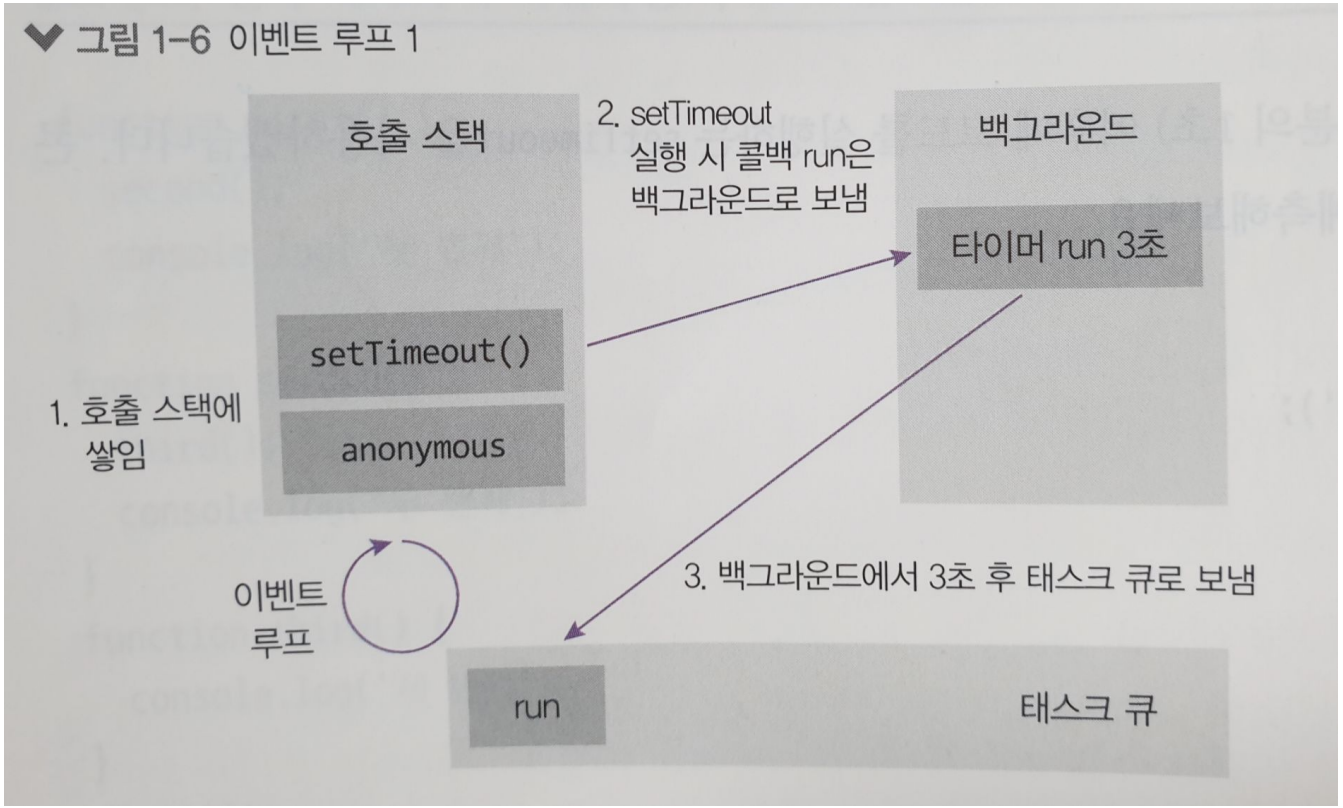


그림 1-6 이벤트 루프 1
호출 스택
2. setTimeout
실행 시 콜백 run은
백그라운드로 보냄
백그라운드
타이머 run 3초
setTimeout()
anonymous
1. 호출 스택에
쌓임
이벤트
루프
3. 백그라운드에서 3초 후 태스크 큐로 보냄
run
태스크 큐

1.1.3 이벤트 루프

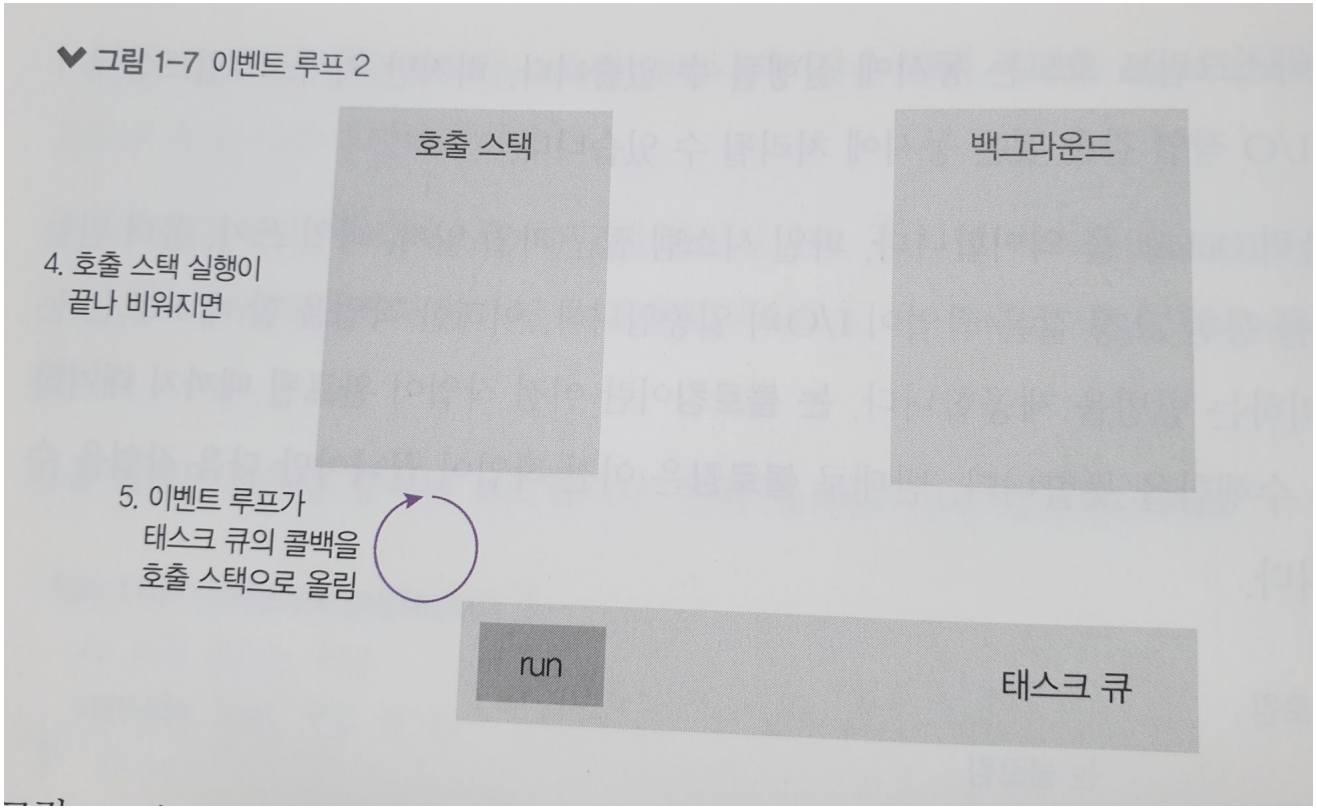


그림 1-7 이벤트 루프 2
호출 스택
백그라운드
4. 호출 스택 실행이
끝나 비워지면
5. 이벤트 루프가
태스크 큐의 콜백을
호출 스택으로 올림
run
태스크 큐

1.1.3 이벤트 루프

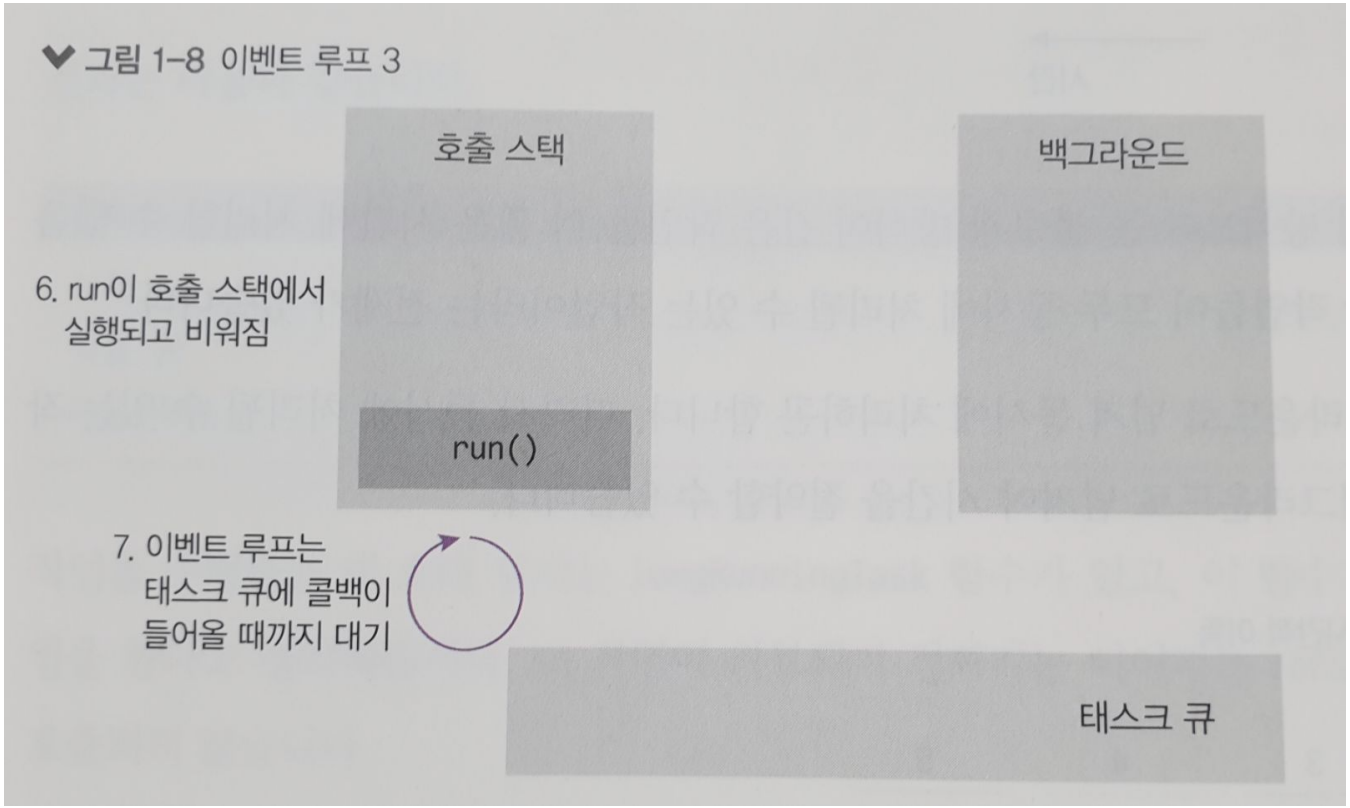


그림 1-8 이벤트 루프 3
호출 스택
백그라운드
6. run이 호출 스택에서
실행되고 비워짐
run()
7. 이벤트 루프는
태스크 큐에 콜백이
들어올 때까지 대기
태스크 큐

1.1.4 논 블로킹 I/O

논 블로킹: 이전 작업이 완료될 때까지 대기하지 않고 다음 작업을 수행

블로킹: 이전 작업이 끝나야만 다음 작업을 수행

1.1.5 싱글 스레드 (프로세스 vs 스레드)

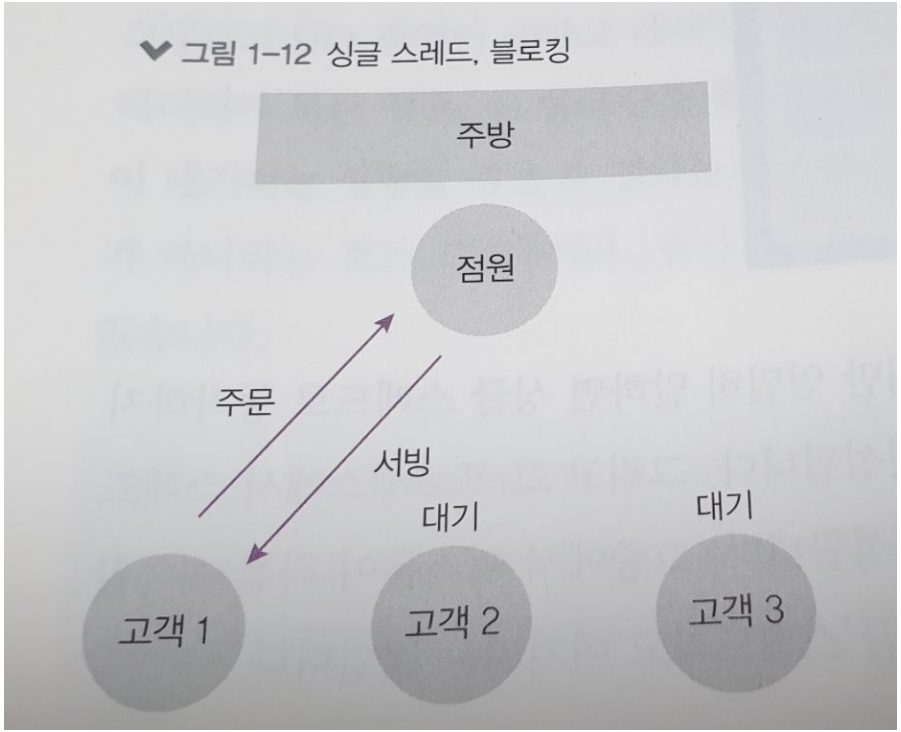
프로세스: 운영체제에서 할당하는 작업의 단위 (노드, 웹 브라우저, 백신 등)

스레드: 프로세스 내에서 실행되는 흐름의 단위. 프로세스와 스레드의 관계는 1:N

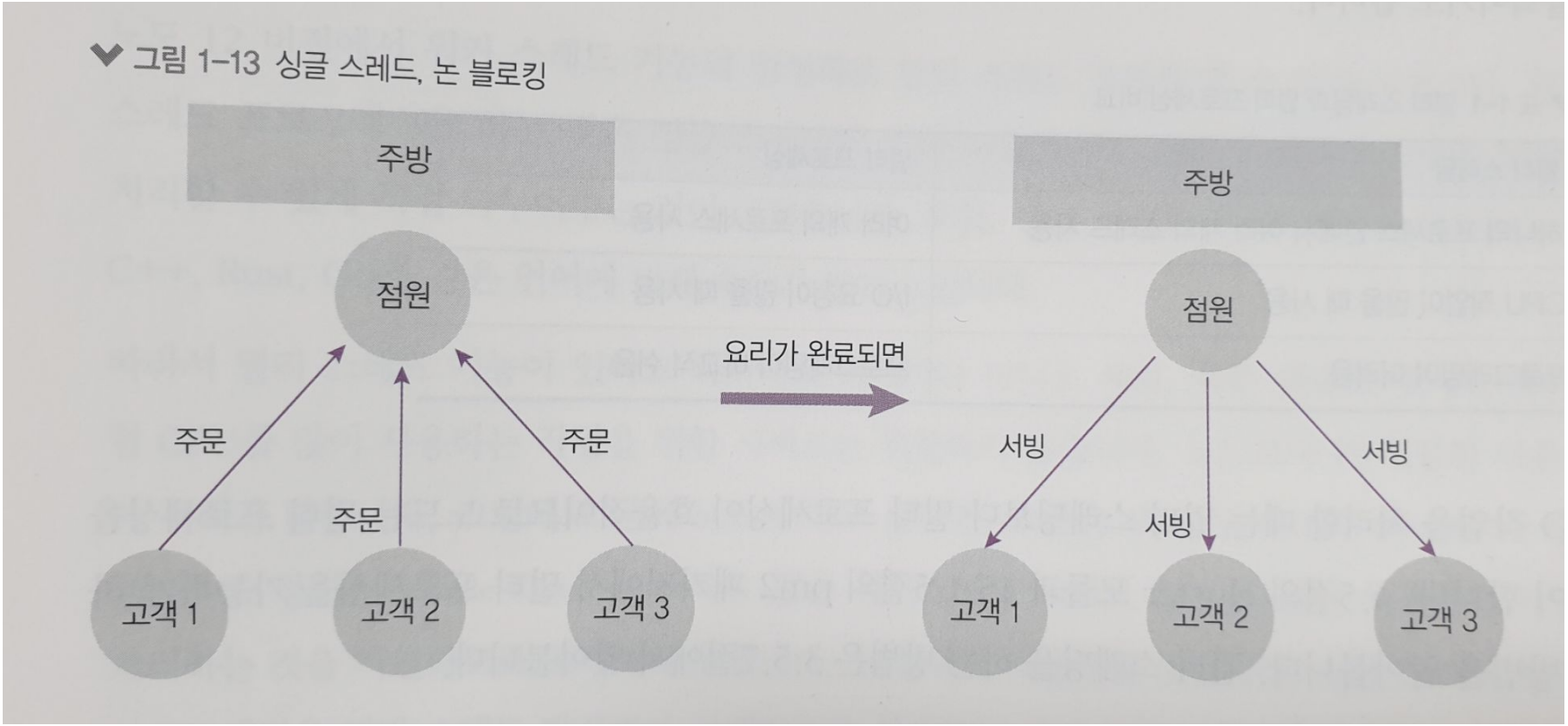
1.1.5 싱글 스레드 (?)

- 사실 노드는 싱글 스레드가 아니다!!?
- 실제로는 여러개 돌지만 우리가 제어 가능한 스레드는 1개이기 때문에 싱글 스레드라고 부르는 것

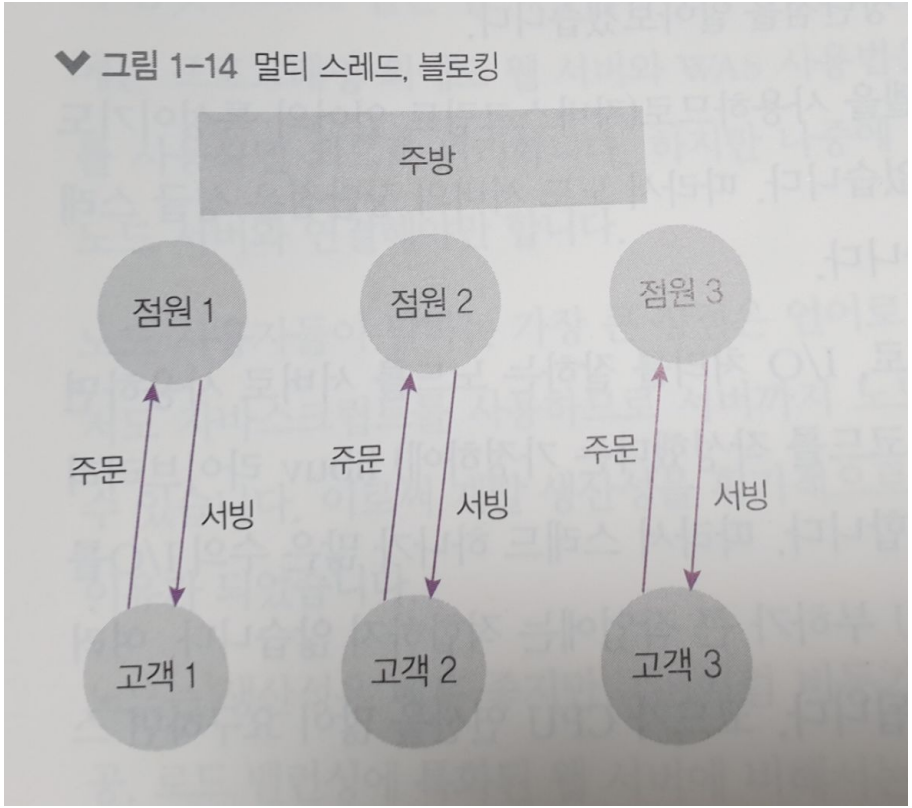
1.1.5 싱글 스레드 (싱글 & 블로킹)



1.1.5 싱글 스레드 (싱글 & 논블로킹)



1.1.5 싱글 스레드 (멀티 & 블로킹)



1.2 서버로서의 노드

- 서버는 I/O 요청이 많기 때문에 노드가 이점에서 유리
- CPU 연산이 많아야하는 서버에는 부적합
- 즉, 가벼운 요청이 많이 발생하는 서버에 적합 (매셥 프로젝트에 적합!)
- **=>** 채팅, 주식 차트
- 노드 12부터 워커 스레드를 통해 멀티 스레드 작업이 가능은하다. 어렵다. 속도 또한 C++, 러스트, Go에 비해 딸린다!
- 다른 대안으로는 서버리스! (AWS Lambda, **Google Cloud Function**)
- 아파치, nginx, Tomcat 바이바이
- **생산성이 좋은 생산성을 위한 생산성 때문에 쓰는 노드 (feat. JS)**

1.3 서버 외의 노드

- 웹, 모바일, 데스크톱 애플리케이션 개발에도 사용
- 웹 프레임워크: 앵귤러, 리액트, 뷰
- 모바일: 리액트 네이티브
- 데스크톱: 일렉트론

1.4 개발 환경 설정하기

- Pass

2장. 알아두어야 할 JS

(feat. 노근본)

2.1 ES2015+

- ES2015 === ES6
- ES2020까지 나옴
- 노드6 버전부터 ES2015 사용 가능

2.1.1 const, let

- const는 수정 불가
- let은 수정 가능

2.1.1 const, let vs var

- const, let: block scope
- var: function scope

2.1.2 템플릿 문자열

- const str = num1 + ‘ 더하기 ‘ + num2 + ‘는 ‘ ……;
- => const str = `${num1} 더하기 ${num2}는 …`;

2.1.3 객체 리터럴

- let es = ‘ES’;
- const obj = {

    [es + 6]: ‘Fantastic’,

  };

2.1.4 화살표 함수

- function add1(x, y) {

  return x + y;

  }
- const add1 = (x, y) => {

  return x + y;

  }

2.1.4 화살표 함수

- 화살표 함수는 상위 스코프의 this를 그대로 물려 받음

2.1.5 구조분해 할당

- const person = {

  name: ‘길동',

  age: 17,

  };

- const { name, age } = person;

2.1.6 클래스

- 프로토타입 기반으로 동작
- 프로토타입 문법을 보기에만 클래스로 보이게 바꾼 것 (Syntactic Sugar)

2.1.7 Promise

- 콜백 패턴의 콜백 지옥 해결사로 등장

2.1.8 Async / Await

- 콜백 패턴의 콜백 지옥 해결사로 등장하였던 Promise의 지옥을 해결하기 위해 등장
- 노드 7.6 버전부터 사용 가능

2.1.8 Promise vs Async / Await

- Promise는 코드를 여러 분기로 나누게 되어 가독성이 안 좋아짐 (개인적인 생각)
- 꼭 분기해야하는 부분이 아니라면 Async / Await...

2.2.1 AJAX

- Asynchronous Javascript And XML
- 요즘은 JSON을 많이 사용
- 페이지 이동 없이 Request/Response 해주는 기술
- jQuery & axios 라이브러리를 많이 사용

2.2.2 FormData

- HTML form 태그의 데이터를 동적으로 제어할 수 있는 기능

2.2.3 encodeURIComponent, decodeURIComponent

- 한글 주소는 encode해서 보내자!
- 서버측에서는 decode

2.2.4 데이터 속성과 dataset

- HTML과 관련된 데이터를 저장하는 공식적인 방법

3장. 노드 기능 알아보기

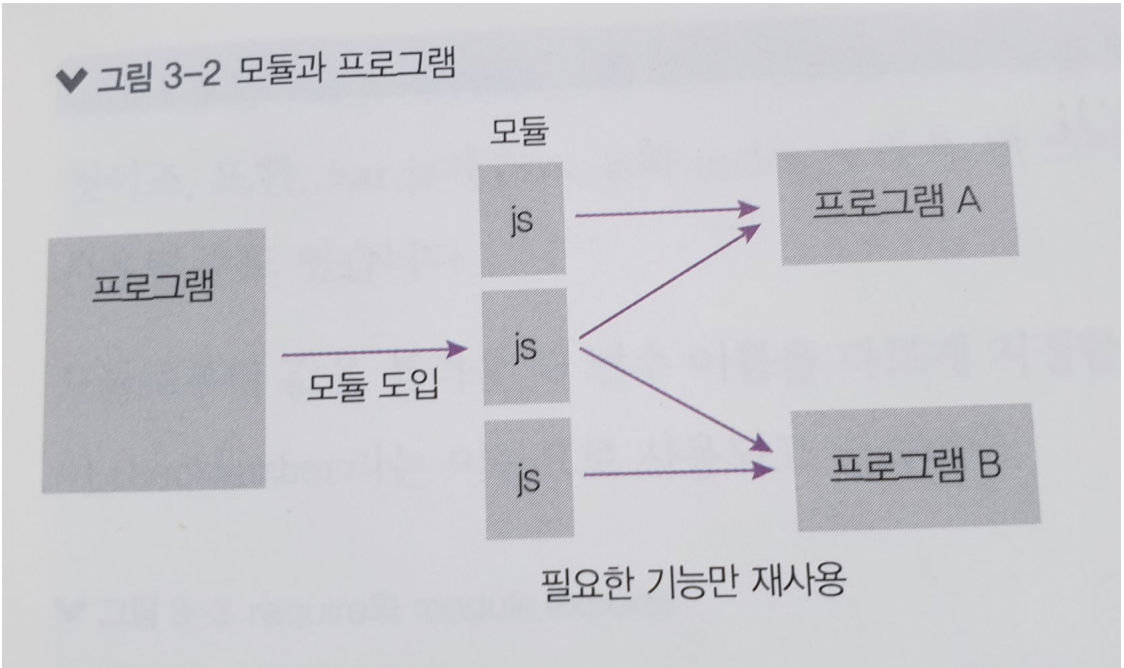
3.1 REPL 사용하기

- Read, Eval, Print, Loop

3.2 JS 파일 실행하기

- 이것은 실습

3.3 모듈로 만들기



3.3 import, export vs require, module.exports

- ?

3.4.1 global

- 전역 객체
- require도 global.require임 (global은 생략 가능)
- global을 통해 서로 다른 js 파일간에 데이터 공유가 가능하지만 …. 헬..

3.4.2 console

- console.log(내용)
- console.time(레이블) & console.timeEnd(레이블)
- console.error(에러 내용)
- console.table(배열)
- console.dir(객체, 옵션)
- console.trace(레이블)

3.4.3 타이머

- setTimeout(콜백 함수, 밀리초): 특정 밀리초 이후에 콜백 함수 실행
- setInterval(콜백 함수, 밀리초): 주어진 밀리초마다 콜백 함수를 반복 실행
- setImmediate(콜백 함수): 콜백 함수를 즉시 실행
- clearTimeout
- clearInterval
- clearImmediate

3.4.3 타이머 (setImmediate vs setTimeout)

- I/O 작업의 콜백 함수 내에서 호출하는 경우 setImmediate가 빠름

3.4.4 __filename, __dirname

- console.log(__filename) => C:/workspace/test.js
- console.log(__dirname) => C:/workspace
- 디테일하게 사용하기위한 path 모듈이 별도로 존재

3.4.5 module, exports, require

- module.exports === exports
- require와 module.exports(exports)가 꼭 최상단에 위치할 필요는 없다!
- require.cache에 캐싱중이기 때문에 여러번 require해도 재사용 된다!
- a모듈에서 b를 부르고, b모듈에서 a를 부르면 순환참조가 발생하는데 객체가 비워져버림!!!!

3.4.6 process

- process.version => v14.0.0 (노드 버전)
- process.arch => x64
- process.platform => win32, linux, darwin, freebsd
- process.pid => 14736
- process.uptime() => 199.36 (프로세스 실행 시간)
- process.execPath => C:/nodejs/node.exe
- process.cwd() => C:/workspace/test
- process.cpuUsage() => { user: 390000, system: 203000 }

3.4.6.1 process.env

- 환경 변수가 저장되어 있음
- 시스템 환경 변수는 노드에 직접 영향을 미치기도 함
- 예) NODE_OPTIONS, UV_THREADPOOL_SIZE
- 보통 API 키나 암호를 .env파일에 저장하고 ‘dotenv’ 모듈을 통해 import하는 방식 사용

3.4.6.2 process.nextTick(콜백)

- setImmediate, setTimeout 보다 빠름
- process.nextTick & Promise를 마이크로 태스크라 부름

3.4.6.3 process.exit

- 0: 정상 종료, 1: 비정상 종료

3.5.1 os

- os.arch() => x86
- os.platform() => win32
- os.type() => Windows_NT
- os.uptime() => 53354
- os.hostname() => DESKTOP-PENGIN
- os.release() => 10.0,18
- os.homedir(), os.timpdir()
- os.cpus().length => 8
- os.freemem(), os.totalmem()

3.5.2 path

- 윈도우와 리눅스에서는 path 생성 방식이 다른데 이러한 문제를 해결해주고 편리하게 파싱하는 기능을 제공

3.5.3 url

- ‘https://www.naver.com/hello/a/b/c’ 을 쉽게 관리해주는 모듈

3.5.5 crypto

- 암호화를 도와주는 모듈
- cpypto 내부의 몇몇 메소드들은 내부적으로 스레드풀을 사용하여 멀티 스레딩으로 동작!

3.5.6 util

- deprecate: deprecated 알림
- promisify: callback pattern => promise pattern

3.5.7 worker_threads

- 멀티 스레딩 프로그래밍을 가능하게 해주는 모듈
- 그러나! 상당히 어려움! 잘못쓰면 싱글 스레드보다 느려짐
- 차라리 프로세스 생성해서 그 친구한테 연산 맡기는게 나을듯..?

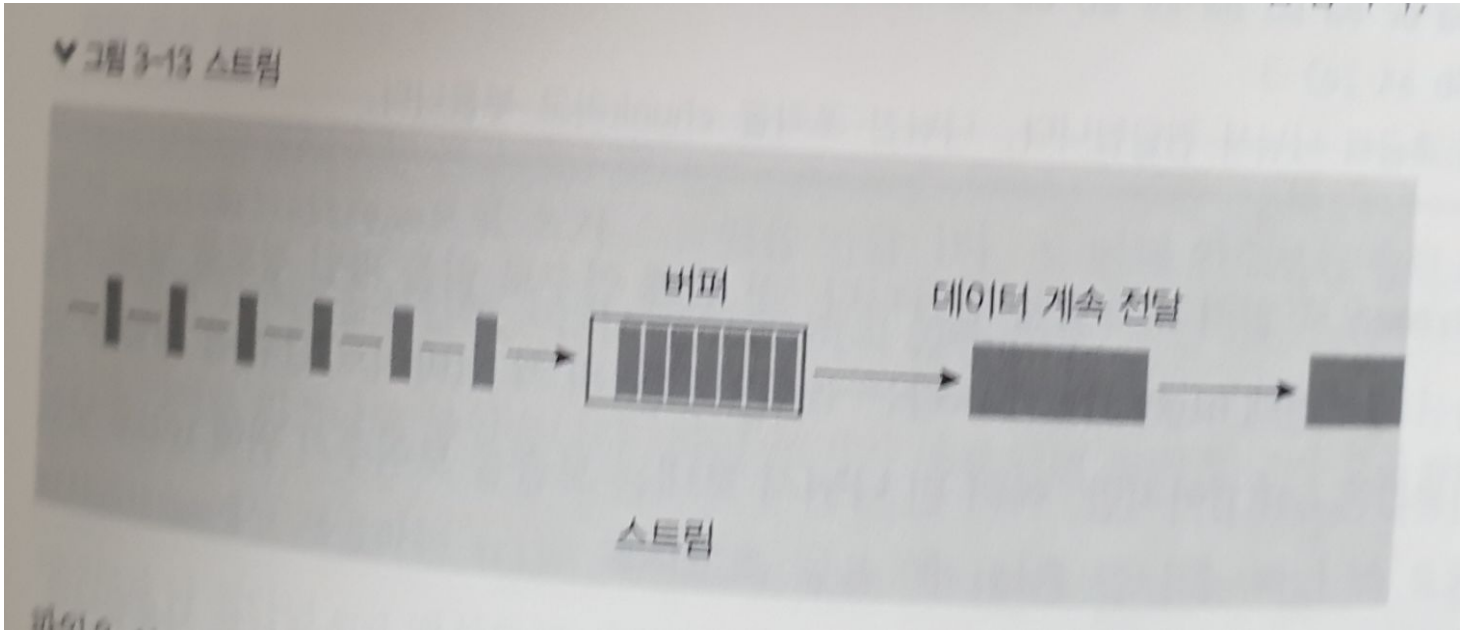
3.5.8 child_process

- 책임 전가하기 편하게 해주는 갓갓 모듈
- 다른 프로세스를 띄워서 명령을 수행하고 결과를 받아올 수 있음
- 예를 들면 파이썬 코드

3.6 파일 시스템 접근하기

- File create/read/update/delete
- 비동기 메소드를 제공함 (일부는 동기 메소드도 제공! readFileSync, writeFileSync)

3.6.2 버퍼 & 스트림



3.6.4 스레드풀 알아보기

- fs, crypto, zlib, dns.lookup에서만 사용
- libuv에서 비동기 작업은 시스템 API로 Wrapping 되어있는 작업은 비동기, 나머지 동기 작업은 쓰레드풀 사용 (https://m.blog.naver.com/pjt3591oo/221976414901)

3.7 이벤트 이해하기

- events 모듈을 통해 나만의 이벤트를 만들어보자!
- on == addListener => 콜백 등록
- emit => 호출
- once => 한 번만 실행되는 이벤트
- removeAllListeners
- removeListener
- off
- listenerCount

3.8 예외 처리하기

- Promise는 알아서 잡아줘서 프로세스가 꺼지지 않음
- Async/Await는 프로세스가 종료 된다!

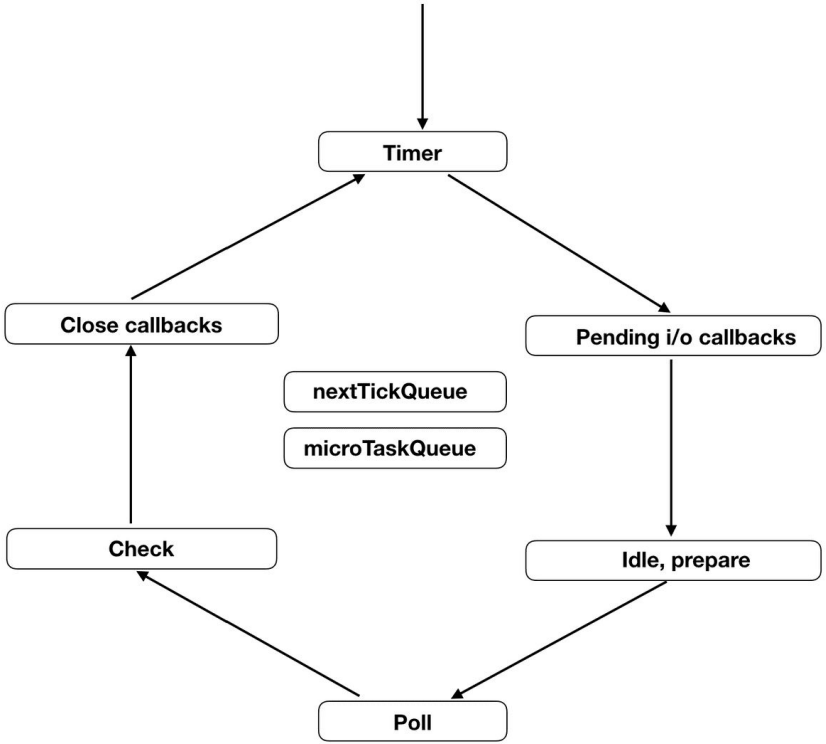
질문

- V8은 뭐하는 녀석이지 => 조금 조사
- import vs require => 10분 세미나
- setImmediate vs setTimeout => 선재
- GC => 10분 세미나(선재?)
- libuv은 누구 => 10분 세미나

이벤트 루프

- 이벤트 루프는 V8 엔진 내부에 있다? => X, 이벤트 루프가 V8 엔진을 자바스크립트 코드를 실행하기 위해 이용하는 것
- 이벤트 루프에서 큐는 하나? => X, 많은 수의 큐로 이루어짐
- 이벤트 루프는 여러 개의 스레드에서 실행? => X, 한 개의 스레드가 자바스크립트 실행 및 이벤트 루프를 담당!

이벤트 루프



- ‘Idle, prepare’ 단계를 빼고 어느 곳에서나 코드 실행 가능
- nextTickQueue, microTaskQueue는 이벤트 루프의 일부가 아니며 어떤 페이지에서든 실행 가능 (가장 높은 우선순위를 가지고 있음)
- Timer에서는 타이머를 min-heap 구조로 관리
- ‘Pending I/O’ 단계에서는 pending_queue에 담겨있는 콜백들을 실행
- ‘Poll’ 단계에서는 watcher_queue에 담겨있는 콜백들을 실행
- ‘Check’단계에서는 `setImmediate()` 를 실행
- ‘Close callbacks’ 단계에서는 .on(‘close’...) 실행