21.8.25 1-1-2 자바스크립트 런타임 ~ 2-1-5

1.1.2 자바스크립트 런타임

공식 사이트의 글을 다시 한 번 보면

> Node.js는 크롬 V8 자바스크립트 엔진으로 빌드된 자바스크립트 런타임입니다. Node.js는 이벤트 기반, 논블로킹 I/O 모델을 사용해 가볍고 효율적입니다. Node.js의 패키지 생태계인 npm은 세계에서 가장 큰 오픈 소스 라이브러리 생태계이기도 합니다.

노드는 자바스크립트 런타임이다. 런타임이란 특정 언어로 만든 프로그램들을 실행할 수 있는 환경을 의미한다.

따라서 노드는 자바스크립트 프로그램을 컴퓨터에서 실행하게 해준다.

▪노드는 V8과 libuv라는 라이브러리를 사용

libuv 라이브러리는 노드의 특성인 이벤트 기반, 논블로킹 I/O 모델을 구현한다.

1.1.3 이벤트 기반

이벤트 기반(event-driven)이란 이벤트가 발생할 때 미리 지정해둔 작업을 수행하는 방식이고 클릭이나 네트워크 요청 등이 있을 수 있다.

따라서 **이벤트 기반 시스템**에서는 특정 이벤트가 발생할 때 무엇을 할지 미리 등록을 해둬야 한다. → 이것을 **이벤트 리스너에 콜백 함수를 등록**한다고 표현함.

만약 발생한 이벤트가 없거나 다 처리하면 다음 이벤트가 발생할 때까지 대기한다.

여러 이벤트가 동시에 발생했을 때 어떤 순서로 콜백 함수를 호출할지는 **이벤트 루프**가 판단한다.

자바스크립트는 코드 첫 줄부터 실행하고 함수 호출 부분을 발견하면 **호출한 함수를 호출 스택**에 넣는다.

```
function first(){
    second();
    console.log('첫번째');
}
function second(){
    third();
    console.log('두번째');
}
function third(){
    console.log('세번째');
}
first();
```

호출 스택은 main() → first() → second() → third() 순서로 쌓이고
실행 되는 순서는 third() → second() → first() → main()이다.

따라서 main까지 실행이 모두 완료 된다면 호출 스택은 비어있게 된다.

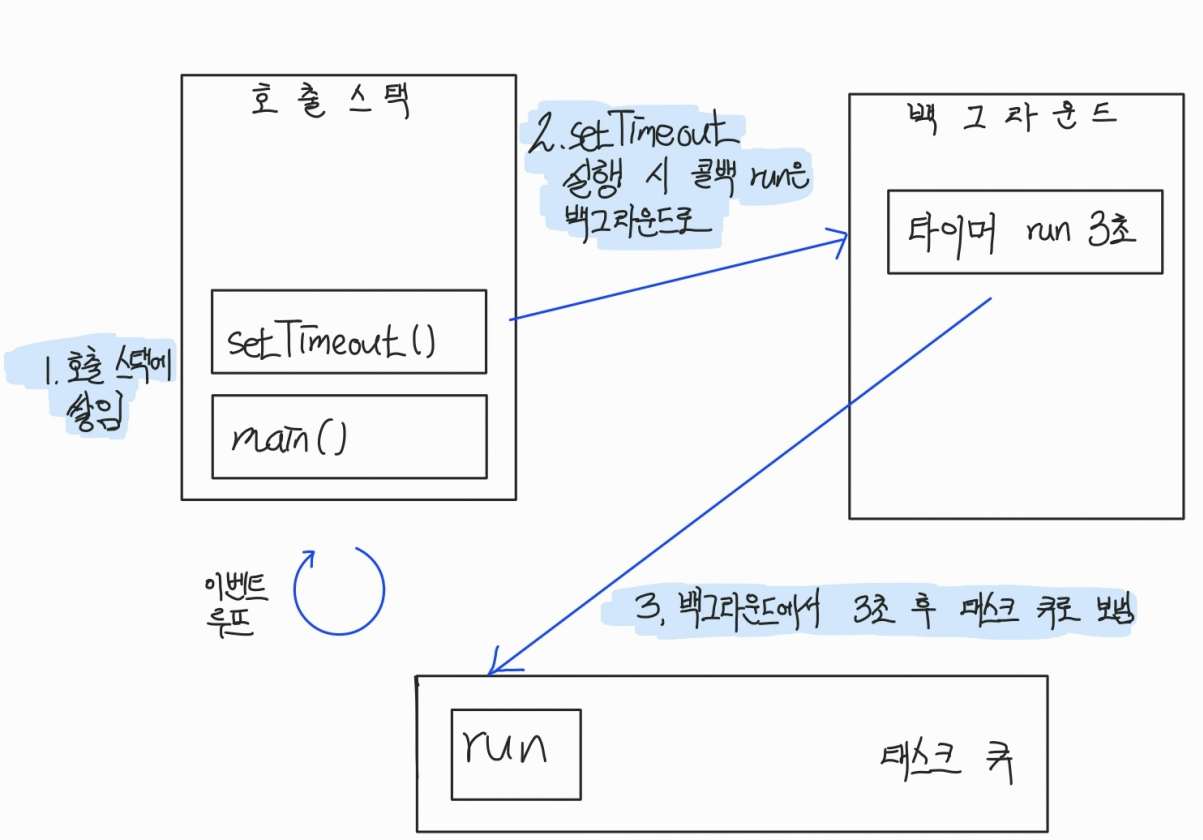

```
function run(){
    console.log('3초 후 실행');
}
console.log('start');

//setTime(a,b) => a를 b(ms)뒤에 실행한다.
setTimeout(run, 3000)
```

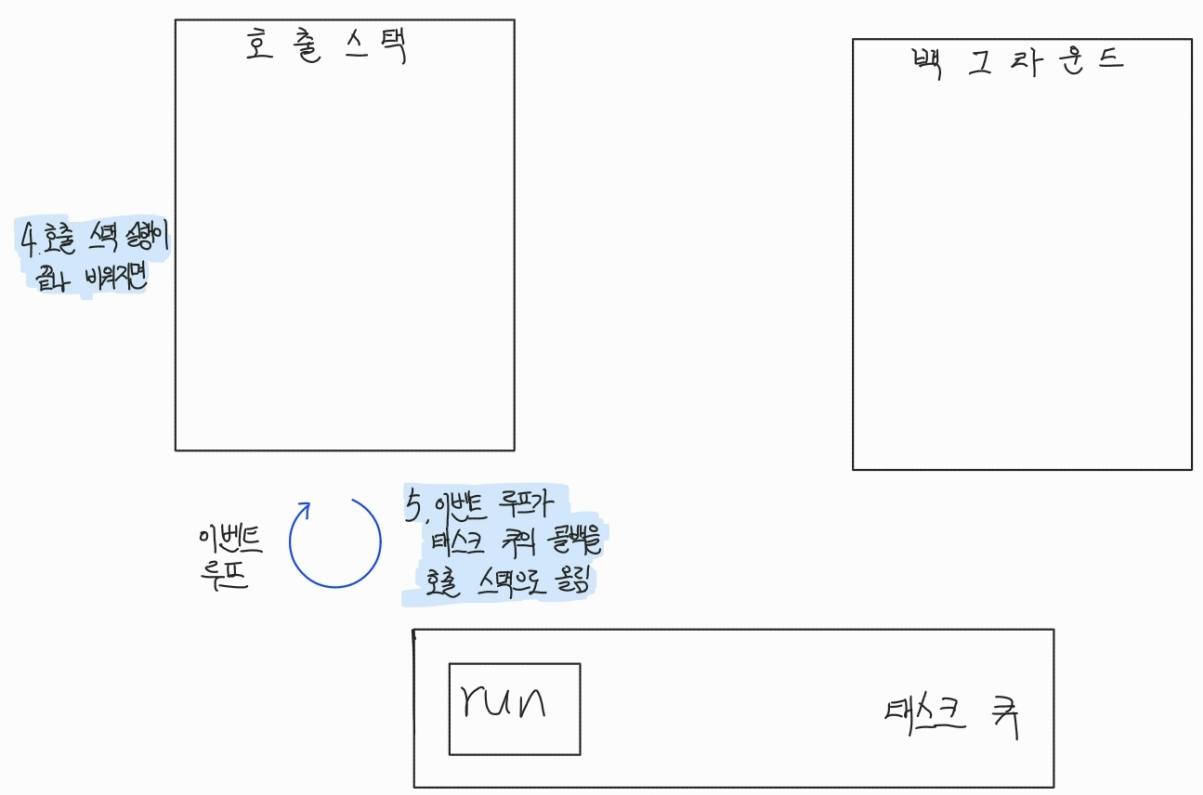
```
console.log('end');
```

```
C:\Program Files\nodejs\node.exe .\21-8-25\callstack2.js
start                                                    callstack2.js:4/21-8-25
end                                                      callstack2.js:8/21-8-25
3초 후 실행                                               callstack2.js:2/21-8-25
```

위 코드를 설명하기 위해서는 이벤트 **루프**, **태스트 큐**, **백그라운드**를 알아야한다.

- **이벤트 루프** : 이벤트 발생 시 호출할 콜백 함수들을 관리, 실행 순서를 결정, 노드가 종료될 때까지 이벤트 처리를 위한 작업을 반복
- **태스크 큐** : 이벤트 발생 후 호출되어야 할 콜백 함수의 대기실, 루프가 정한대로 줄을 서 있는다.
- **백그라운드** : 타이머나 I/O 작업 콜백 또는 이벤트 리스너들이 대기하는 곳.





⚠️ 호출 스택에 함수가 너무 많이 차 있으면 3초가 지나도 함수가 실행되지 않을 수 있다. → **setTimeout의 시간이 정확하지 않을 수 있다.**

1.1.4 논블로킹 I/O

논블로킹 방식 : 이전 작업이 완료될 때까지 멈추지 않고 다음 작업을 수행함

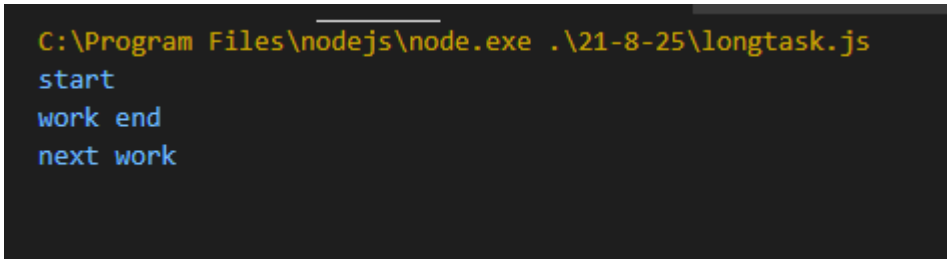
하지만 **싱글 스레드라는 한계** 때문에 자바스크립트의 모든 코드가 시간적 이득을 보진 않음
→ 대신 노드 프로세스 이외에 **다른 컴퓨팅 자원을 사용**할 수 있는 **I/O 작업이 시간적 이득**

만약 오래 걸리는 작업이 있었을 때

```
function longRunningTask() {
    //오래 걸리는 작업
    console.log('work end');
```

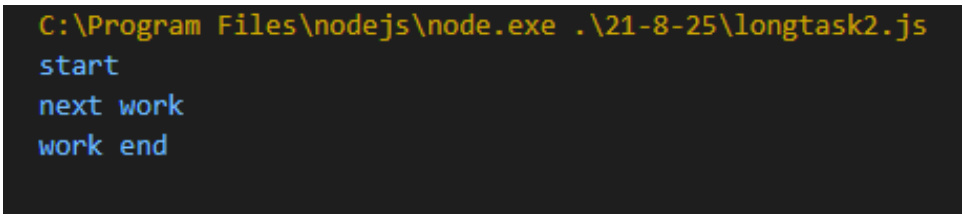
```
}

console.log('start');
longRunningTask();
console.log('next work')
```

이렇게 하면 오래 걸리는 작업부터 실행이 되고 다음 작업이 실행된다.



이번에는 setTimeout을 이용해서 코드를 바꾼다.

```
function longRunningTask() {
    //오래 걸리는 작업
    console.log('work end');
}

console.log('start');
setTimeout(longRunningTask,0);
console.log('next work')
```



setTimeout함수를 쓰게 되면 longRunningTask를 백그라운드에 넣고 태스크큐를 거쳐 호출 스택으로 오는 시간에 next work가 먼저 실행된다.

이 예제는 노드의 논블로킹 방식을 보여주는 좋은 예이다.

1.1.5 싱글 스레드

자바스크립트와 노드에서 논블로킹이 중요한 이유는 싱글 스레드이기 때문이다.

한 번에 한 가지 일밖에 처리하지 못하므로 블로킹이 발생하면 다음 일을 처리하지 못한다.

싱글 스레드, 블로킹 모델 → 하나씩 처리해야하므로 굉장히 비효율적

싱글 스레드, 논블로킹 모델 → 한 스레드가 한번에 여러 일을 실행할 수 있지만 그 스레드가 제 역할을 못한다면 마비되고 일이 많으면 버거울 수 있다.

멀티 스레드, 블로킹 모델 → 일을 잘 처리할 수 있지만 일이 많을수록 스레드도 많아진다. 일이 없을 땐 놀고 있는 스레드가 생길 수 있다.

하지만 노드는 멀티 스레드랑 비슷한 멀티 프로세스 기능을 구현할 수 있다.

> 💡 **프로세스** : 운영체제에서 할당하는 작업의 단위, 프로세스 간 자원 공유X
> **스레드** : 프로세스 내에서 실행되는 흐름의 단위, 부모 프로세스의 자원 공유

1.2 서버로서의 노드

▪노드 서버는 I/O가 많은 작업에 적합하다.(libuv라이브러리를 사용해 논블로킹 방식으로 처리)

하지만 CPU의 코어를 하나 밖에 사용하지 못하기 때문에 코드가 CPU연산을 많이 요구하면 블로킹이 발생

노드의 가장 큰 장점은 자바스크립트를 사용한다는 것 → 하나의 언어로 웹 사이트를 개발 →개발 생산성 증가

▪장점

- 멀티 스레드 방식에 비해 컴퓨터 자원을 적게 사용함
- I/O 작업이 많은 서버로 적합

- 멀티 스레드 방식보다 쉬움
- 웹 서버가 내장되어 있음
- 자바스크립트를 사용함
- JSON 형식과 호환하기 쉬움

▪단점

- 싱글 스레드라서 CPU 코어를 하나만 사용함
- CPU 적업이 많은 서버로는 부적합
- 하나뿐인 스레드가 멈추지 않도록 관리해야 함
- 서버 규모가 커졌을 때 서버를 관리하기 어려움
- 어중간한 성능

이와 같은 특성을 활용하려면 **개수는 많지만 크기는 작은 데이터를 실시간으로 주고 받는 데 적합**.

ex) 실시간 채팅 어플, 주식 차트, JSON 데이터를 제공하는 API 서버

1.3 서버 외의 노드

처음에는 노드를 서버로 사용했지만, 자바스크립트 런타임이므로 서버로 한정되진 않는다.

노드 기반으로 돌아가는 웹 프레임워크는 **Angular**, **React**, **Vue**, **Meteor** 등이 있다.

모바일 개발 도구로는 **React Native**, **Ionic Framework**를 많이 사용한다.

1.4 설치

```
sudo apt-get update
sudo apt-get install -y build-essential
sudo apt-get install curl
```

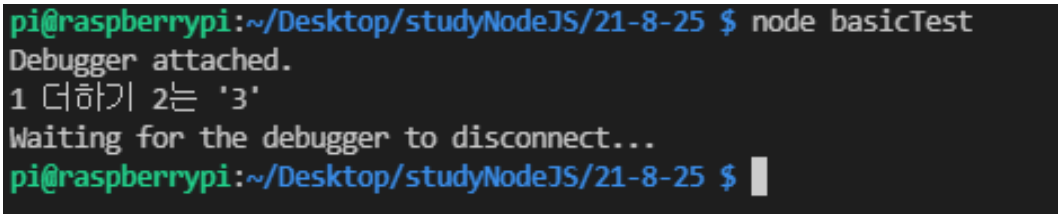
```
curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash --
sudo apt-get install -y nodejs
```

우분투 기준이고 해당 코드만 복붙하면 자동으로 설치 된다.

2.1 ES2015+

템플릿 문자열

```
const num3=1;
const num4=2;
const result2 = 3;
//''나 ""를 묶을 때는 `(백틱)을 사용한다.
var string = `${num3} 더하기 ${num4}는 '${result2}'`;
console.log(string);
```



객체 리터럴

```
var sayNode = function(){
    console.log('Node');
};
var es = 'ES'
const newObject = {
    sayJS(){
        console.log('JS');
    },
    sayNode,
    [es + 6]: 'Fantastic',
};
newObject.sayNode();
newObject.sayJS();
console.log(newObject.ES6);
```

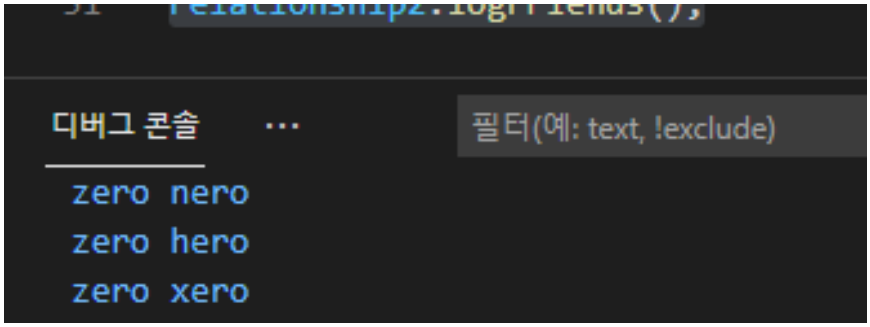
```
pi@raspberrypi:~/Desktop/studyNodeJS/21-8-25 $ node basicTest
Debugger attached.
Node
JS
Fantastic
Waiting for the debugger to disconnect...
```

화살표 함수

```
function add(x,y){
    return x+y;
};
const add2=(x,y)=> {
    return x+y;
};
const add3 = (x,y) => (x+y);

function not1(x){
    return !x;
}
const not2 = x=> !x;
```

<this 활용>

```
const relationship2 = {
    name: 'zero',
    friends: ['nero', 'hero', 'xero'],
    logFriends(){
        this.friends.forEach(friend => {
            console.log(this.name, friend);
        });
    },
};

relationship2.logFriends();
```



🔥🔥🔥비구조화 할당 🔥🔥🔥

```
const candyMachine = {
    status: {
        name : 'node',
        count : 5,
    },
    getCandy : function() {
        this.status.count--;
        return this.status.count;
    }
};
const {getCandy, status: {count}} = candyMachine;
```

아직도 잘 이해되지 않는 구문이다. 지금까지 내가 이해하고 있는 느낌은 현재 있는 값 그대로 가져온다는 느낌이다.

처음에는 저 요소들 자체를 떼와 동기화가 되면서 간추린 형태로 쓸 수 있는 형태인줄 알았지만 테스트를 해보니 아니였다.

일단 값을 추출한다는 형태로 알고 있으려 한다.