1. 노드 시작하기

1.1 핵심 개념 이해하기

1.1.1 서버

- 서버 : 네트워크를 통해 클라이언트에 정보나 서비스를 제공하는 컴퓨터 또는 프로그램

- 클라이언트 : 요청을 보내는 주체로 브라우저일 수도 있고, 데스크톱 프로그램일 수도 있고,   
모바일 앱일 수도 있고, 다른 서버에 요청을 보내는 서버일 수도 있음.

- 서버라고 해서 요청에 대한 응답만 하는 것은 아님. 다른 서버에 요청을 보낼 수도 있으며,   
이때는 요청을 보낸 서버가 클라이언트 역할을 함.

1.1.2 자바스크립트 런타임

- 노드는 자바스크립트 런타임이다. (자바스크립트 실행기)

- 런타임은 특정 언어로 만든 프로그램들을 실행할 수 있는 환경

- 노드는 V8과 더불어 libuv라는 라이브러리를 사용  
(libuv 라이브러리는 노드의 특성인 이벤트 기반, 논 블로킹 I/O 모델을 구현하고 있음)

1.1.3 이벤트 기반

- 이벤트 기반(event-driven)이란, 이벤트가 발생할 때 미리 저장해둔 작업을 수행하는 방식

- 이벤트 기반 시스템에서는 특정 이벤트가 발생할 때 무엇을 할지 미리 등록해두어야 함.  
이를 이벤트 리스너(event listener)에 콜백(callback) 함수를 등록한다고 표현

- 노드도 이벤트 기반 방식으로 동작하므로, 이벤트가 발생하면 이벤트 리스너에 등록해둔 콜백 함수를 호출. 발생한 이벤트가 없거나 발생했던 이벤트를 다 처리하면, 노드는 다음 이벤트가 발생할 때까지 대기

- 이벤트 루프 : 이벤트 발생 시 호출할 콜백 함수들을 관리, 호출된 콜백 함수의 실행 순서 결정

- 백그라운드 : setTimeout 같은 타이머나 이벤트 리스너들이 대기하는 곳

- 테스크 큐 (콜백 큐) : 이벤트 발생 후, 백그라운드에서는 테스크 큐로 타이머나 이벤트 리스너들의 콜백 함수들을 보냄

1.1.4 논 블로킹 I/O

- 논 블로킹 : 이전 작업이 완료될 때까지 대기하지 않고 다음 작업을 수행함  
(논 블로킹과 동시가 같은 의미가 아님)

블로킹 : 이전 작업이 끝나야만 다음 작업을 수행함

- 노드에서는 동기와 블로킹이 유사, 비동기와 논 블로킹이 유사

1.1.5 싱글 스레드

- 노드가 채택하고 있는 방식 : 싱글 스레드, 논 블로킹 모델

- 노드가 싱글 스레드로 동작하지 않는 경우 : 스레드풀, 워커 스레드

1.2 서버로서의 노드

- 노드의 장점 : 멀티 스레드 방식에 비해 적은 컴퓨터 자원 사용, I/O 작업이 많은 서버로 적합, 멀티 스레드 방식보다 쉬움, 웹 서버가 내장되어 있음, 자바스크립트를 사용함, JSON 형식과 쉽게 호환됨

- 노드의 단점 : 기본적으로 싱글 스레드라서 CPU 코어를 하나만 사용, CPU 작업이 많은 서버로는 부적합, 하나뿐인 스레드가 멈추지 않도록 관리가 필요함, 서버 규모가 커졌을 때 서버를 관리하기 어려움, 어중간한 성능

1.3 서버 외의 노드

- 노드 기반으로 돌아가는 대표적인 웹 프레임워크 : 앵귤러(Angular), 리액트(React), 뷰(Vue) 등

1.4 개발 환경 설정하기

- 노드 설치

- npm 버전 업데이트

- 비주얼 스튜디오 코드 설치

2. 알아두어야 할 자바스크립트

2.1 ES2015+

2.1.1 const, let

- 보통 자바스크립트를 배울 때는 var로 변수를 선언하는 방법부터 배움. 하지만 var은 이제 const와 let이 대체함.

- const와 let의 차이 : const는 한 번 값을 할당하면 다른 값을 할당할 수 없음, 초기화할 때 값을 할당하지 않으면 에러 발생 (상수)

2.1.2 템플릿 문자열

- 백틱( ` ) : 문자열 안에 변수를 넣을 수 있음

2.1.3 객체 리터럴

- ES5 : { name: name, age: age }

- ES2015 : { name, age }

2.1.4 화살표 함수

- 화살표 함수에서는 function 선언 대신 => 기호로 함수를 선언함. 또한 변수에 대입하면 나중에 재사용 가능

2.1.5 구조분해 할당

- 구조분해 할당을 사용하면 객체와 배열로부터 속성이나 요소를 쉽게 꺼낼 수 있음.

2.1.6 클래스

- 다른 언어처럼 클래스 기반으로 동작하는 것이 아니라, 자바스크립트는 여전히 프로토타입 기반으로 동작함.

2.1.7 프로미스

- 자바스크립트와 노드에서는 주로 비동기를 접함 (특히 이벤트 리스너를 사용할 때 콜백 함수를 자주 사용)

- 프로미스를 쉽게 설명하자면, 실행은 바로 하되 결과값은 나중에 받는 객체 (결과값은 실행이 완료된 후 then이나 catch 메서드를 통해 받는다)

- 프로미스를 return한 경우에는 프로미스가 수행된 후 다음 then이나 catch가 호출된다.

- 프로미스 여러 개를 한 번에 실행하는 방법 : Promise.all

2.1.8 async/await

- 노드 7.6 버전부터 지원되는 기능, ES2017에서 추가됨

- 함수 선언부를 일반 함수 대신 async function으로 교체한 후, 프로미스 앞에 await를 붙임

- for await of : 프로미스 배열을 순회할 수 있음

- async 함수의 반환값은 항상 Promise로 감싸짐

2.2 프런트엔드 자바스크립트

2.2.1 AJAX (Asynchronous Javascript And XML)

- 비동기적 웹 서비스를 개발할 때 사용하는 기법

- 이름에 XML이 들어 있지만 꼭 XML을 사용해야 하는 것은 아니며, 요즘에는 JSON을 많이 사용

- 쉽게 말해 페이지 이동 없이 서버에 요청을 보내고 응답을 받는 기술

- 웹 사이트 중에서 페이지 전환 없이 새로운 데이터를 불러오는 사이트는 대부분 AJAX 기술을 사용하고 있다고 보면 됨

- AJAX 요청 : jQuery나 axios 같은 라이브러리를 이용해서 보냄

2.2.2 FormData

- HTML form태그의 데이터를 동적으로 제어할 수 있는 기능 (주로 AJAX와 함께 사용)

- append 메서드 : 키-값 형식의 데이터를 저장할 수 있음, has 메서드 : 주어진 키에 해당하는 값이 있는지의 여부를 알림, get 메서드 : 주어진 키에 해당하는 값 하나를 가져옴, getAll : 해당하는 모든 값을 가져옴, delete 메서드 : 현재 키를 제거, set : 현재 키를 수정

2.2.3 encodeURIComponent, decodeURIComponent

- AJAX 요청을 보낼 때, 주소에 한글이 들어가는 경우

- 서버 종류에 따라 다르지만 서버가 한글 주소를 이해하지 못하는 경우가 있는데, 이럴 때는 window 객체의 메서드인 encodeURIComponent 메서드를 사용

- 받는 쪽에서는 decodeURIComponent를 사용

2.2.4 데이터 속성과 dataset

- 보안 : 프런트엔드에서 데이터를 내려보낼 때 첫 번째로 고려해야 할 점임. (클라이언트를 믿지 마라!)

- 데이터 속성(data attribute) : HTML5에서 HTML과 관련된 데이터를 저장하는 공식적인 방법

3. 노드 기능 알아보기

3.1 REPL 사용하기

- 자바스크립트는 스크립트 언어이므로 미리 컴파일을 하지 않아도 즉석에서 코드를 실행할 수 있음.

- REPL(Read Eval Print Loop) : 입력한 코드를 읽고(Read), 해석하고(Eval), 결과물을 반환하고(Print), 종료할 때까지 반복한다(Loop)

- 노드의 REPL을 직접 사용하기 : 윈도우에서는 명령 프롬프트, 맥이나 리눅스에서는 터미널을 열고 node를 입력, VS Code에서는 Ctrl + ` 을 누르면 터미널을 켤 수 있음

- REPL 종료 : Ctrl + C 를 두 번 누르거나, REPL 창에 .exit을 입력

- REPL은 한두 줄짜리 코드를 테스트해보는 용도로는 좋지만 여러 줄의 코드를 실행하기에는 불편함. 긴 코드인 경우에는 코드를 자바스크립트 파일로 만든 후 파일을 통째로 실행하는 것이 좋음.

3.2 JS 파일 실행하기

- 자바스크립트 파일을 생성

- 콘솔에서 node [자바스크립트 파일 경로]로 실행. 확장자(.js)는 생략해도 됨

3.3 모듈로 만들기

- 노드는 코드를 모듈로 만들 수 있다는 점에서 브라우저의 자바스크립트와 다르다.

- 모듈이란 특정한 기능을 하는 함수나 변수들의 집합, 모듈 자체로도 하나의 프로그램이면서 다른 프로그램의 부품으로도 사용할 수 있음.

- 모듈로 만들어두면 여러 프로그램에 해당 모듈을 재사용할 수 있음

3.4 노드 내장 객체 알아보기

3.4.1 global

- 브라우저의 window와 같은 전역 객체 (전역 객체 : 모든 파일에서 접근하 수 있음)

- 생략 가능 (global.require -> require, global.console -> console)

- 노드의 window, document 객체 : 노드에 DOM이나 BOM이 없으므로 window나 document 객체는 노드에서 사용할 수 없음.

- global 객체의 남용 : global 객체의 속성에 값을 대입하여 파일 간에 데이터를 공유할 수 있지만, 이를 남용해서는 안 됨. 프로그램의 규모가 커질수록 어떤 파일에서 global 객체에 값을 대입했는지 찾기 힘들어져 유지 보수에 어려움을 겪을 수 있음. 다른 파일의 값을 사용하고 싶다면, 모듈 형식으로 만들어서 명시적으로 값을 불러와 사용하는 것이 좋음.

3.4.2 console

- console도 노드에서는 window 대신 global 객체 안에 들어 있음

- console 객체는 보통 디버깅을 위해 사용, 대표적으로 console.log 메서드 (개발하면서 변수에 값이 제대로 들어 있는지 확인할 때, 에러 발생 시 에러 내용을 콘솔에 표시하기 위해, 코드 실행 시간을 알아보려고 할 때)

- 다른 로깅 함수들

\* console.time(레이블) : console.timeEnd(레이블)과 대응되어 같은 레이블을 가진 time과 timeEnd 사이의 시간을 측정

\* console.log(내용) : 평범한 로그를 콘솔에 표시함. console.log(내용, 내용, …)처럼 여러 내용을 동시에 표시할 수도 있음.

\* console.error(에러 내용) : 에러를 콘솔에 표시

\* console.table(배열) : 배열의 요소로 객체 리터럴을 넣으면, 객체의 속성들이 테이블 형식으로 표현됨.

\* console.dir(객체, 옵션) : 객체를 콘솔에 표시할 때 사용함. 옵션의 colors를 true로 하면 콘솔에 색이 추가되어 보기가 한결 편해짐, depth는 객체 안의 객체를 몇 단계까지 보여줄지를 결정함. (기본값 : 2)

\* console.trace(레이블) : 에러가 어디서 발생했는지 추적할 수 있게 함.

3.4.3 타이머

- 타이머 기능을 제공하는 함수인 setTimeout, setInterval, setImmediate는 노드에서 window 대신 global 객체 안에 들어 있음.

\* setTimeout(콜백 함수, 밀리초) : 주어진 밀리초(1,000분의 1초) 이후에 콜백 함수를 실행

\* setInterval(콜백 함수, 밀리초) : 주어진 밀리초마다 콜백 함수를 반복 실행

\* setImmediate(콜백 함수) : 콜백 함수를 즉시 실행

- 이 타이머 함수들은 모두 아이디를 반환함. 아이디를 사용하여 타이머를 취소할 수 있음.

\* clearTimeout(아이디) : setTimeout을 취소

\* clearInterval(아이디) : setInterval을 취소

\* clearImmediate(아이디) : setImmediate를 취소

- cf) setImmediate(콜백)과 setTimeout(콜백, 0)

\* 파일 시스템 접근, 네트워킹 같은 I/O 작업의 콜백 함수 안에서 타이머를 호출하는 경우는setImmediate가 setTimeout(콜백, 0)보다 먼저 실행됨.

\* 하지만, setImmediate가 항상 setTimeout(콜백, 0)보다 먼저 호출되지는 않음.

\* setTimeout(콜백, 0)은 사용하지 않는 것을 권장

3.4.4 \_\_filename, \_\_dirname

- 파일에 \_\_filename과 \_\_dirname을 넣어두면, 실행 시 현재 파일명과 현재 파일 경로로 바뀜.

3.4.5 module, exports, require

- module.exports : 한 번에 대입

- 각각의 변수를 exports 객체에 하나씩 넣는 방법

- 한 번 require한 파일은 require.cache에 저장되므로, 다음 번에 require할 때는 새로 불러오지 않고 require.cache에 있는 것이 재사용됨 / require.main : 노드 실행 시 첫 모듈을 가리킴

3.4.6 process

- process 객체는 현재 실행되고 있는 노드 프로세스에 대한 정보를 담고 있음

3.4.6.1 process.env

- 시스템의 환경 변수를 출력 (왼쪽이 환경 변수의 이름, 오른쪽이 값)

NODE\_OPTIONS=--max-old-space-size=8192

UV\_THREADPOOL\_SIZE=8

\* NODE\_OPTIONS : 노드를 실행할 때의 옵션들을 입력받는 환경 변수

\* --max-old-space-size=8192 : 노드의 메모리를 8GB까지 사용할 수 있게 함

\* UV\_THREADPOOL\_SIZE : 노드에서 기본적으로 사용하는 스레드풀의 스레드 개수를 조절할   
 수 있게 함

- process.env는 서비스의 중요한 키를 저장하는 공간으로도 사용됨

3.4.6.2 process.nextTick(콜백)

- 이벤트 루프가 다른 콜백 함수들보다 nextTick의 콜백 함수를 우선으로 처리하도록 만듦

- process.nextTick은 setImmediate나 setTimeout보다 먼저 실행됨

- 코드 맨 밑에 Promise를 넣은 것은 resolve된 Promise도 nextTick처럼 다른 콜백들보다 우선시되기 때문. 그래서 process.nextTick과 Promise를 마이크로태스크(microtask)라고 따로 구분지어 부름

3.4.6.3 process.exit(코드)

- 실행 중인 노드 프로세스를 종료

- 서버 환경에서 이 함수를 사용하면 서버가 멈추므로, 특수한 경우를 제외하고는 서버에서 잘 사용하지 않음. 하지만, 서버 외의 독립적인 프로그램에서는 수동으로 노드를 멈추기 위해 사용

- process.exit 메서드는 인수로 코드 번호를 줄 수 있음. 인수를 주지 않거나 0을 주면 정상 종료를 뜻하고, 1을 주면 비정상 종료를 뜻함 (만약 에러가 발생해서 종료하는 경우에는 1을 넣으면 됨)