Node.js 02 [자바스크립트 기본]

프론트엔드 개발환경의 이해와 실습

조현영

2022.09.18

호출 스택

함수 호출할 때 함수 내에서 다른 함수가 실행될시에 스택을 쌓아 FILO 형식으로 실행시킨다.

(실행시킨다 라는 표현보다는 실행을 완료시키고 스택에서 나간다가 맞는 표현)

그렇다면 **setTimeout** 사용시**?**

비동기를 해결하기 위해 이벤트 루프를 알아야 한다.

이벤트 루프

setTimeout (run, x초) 함수가 진행된다고 생각해보자

비동기 함수가 실행된다면 백그라운드에 타이머 (run , x초) 등을 생성해 준다. 코드가 백그라운드로 진행되면 호출스택과 동시에 진행된다. (백그라운드에서 실행되는 것)

-백그라운드는 다른 스레드에서 진행된다. [백그라운드로 보낼 수있는 관련 함수는 정해져 있다]

나머지 코드가 진행되고 - 백그라운드 타이머가 완료되면 태스크 큐로 run함수를 이동시켜주고

태스크 큐의 코드(run) 을 호출 스택으로 옮겨서 실행된다.

[[setTimeout(run, 0초) -바로 실행X → 백그라운드로 이동한다.

- 호출 스택 영역이 자바스크립트 이고 백그라운드, 태스크 큐는 다른 언어로 설계되어 멀티 스레드를 사용하는 것.

const, let

ES2015 이전에는 var 변수를 선언 → 이후 const, let

가장 큰 차이점은 스코프 (var은 함수 스코프)

var은 블록을 무시한다. (function() 스코프)

const

- const a = 3; 한번 만 선언 가능 (객체 멤버를 변경을 가능)
- **' = '** 을 단 한번만 사용 가능하다**.**

let

- 변경 할 수 있다.
- ' = ' 여러번 설정 가능 그러므로 주로 const 로 선언 후 필요시 let 쓰는방식 선호 됨

탬플릿 형식

const result = `a의 값은 ${a}입니다.`;

백킹 문자 ~ –'``'

function a(){}

a(); 호출

a``; 호출

화살표 함수

function add1(x, y) { return x + y ; }

⇒ const add2 = (x, y) => { return x + y;}; [중괄호 { 다음 return이 나오면 생략가능]

⇒ const add3 = (x, y) => x + y ;

⇒ const add4 = (x, y) => (x + y);

화살표 함수 **function**

this를 사용하는 것에 차이,

this를 사용하면 function을 사용하는 것이 좋다.

구조분해 문법

const example = { a: 123, b: { c : 135, d: 146 } }

const a = example.a;

const d = example.b.d;

⇒ **const { a, b : { d } } = example;** 구조는 키를 맞춰서 사용

arr = [1, 2, 3, 4, 5]

const x = arr[0]

const y = arr[1]

const z = arr[4]

⇒ **const[x,y, , , z] = arr;** 배열은 자리수

주의점 **: this** 사용시 구조분해 할당시 오류 발생 할 수 있다**.**

클래스

클래스는 프로토 타입 . 프로토 타입을 깔끔하게 만든것이 클래스

Class문법 예시

```
class Human{
    constructor(type = 'human'){
        this.type = type;
    }
    static isHuman(human){
        return human instanceof Human;
    }
    breathe(){
        alert('h-a-a-m');
    }
}
```

상속

```
class Zero extends Human{
    constructor(type, firstName, lastName){
        super(type);
        this.firstName = firstName;
        this.lastName = lastName;
    }
    sayName(){
        super.breathe();
        alert(`${this.firstName} ${this.lastName}`);
    }
}
```

프로미스

프로미스 : 내용이 실행은 되었지만 결과를 아직 반환하지 않은 객체

Then을 붙이면 결과를 반환함 – 실행이 완료되지 않았으면 완료된 후에 Then 내부 함수가 실행된다.

Callback 의 차이는 코드가 분리될 수 있는 지의 여부다.

프로미스 변수에 담아두고 then으로 코드를 분리할 수 있다.

resolve → 성공 리턴값 **then**

reject → 실패 리턴값 **catch**

Promise→ all 보다 allSettled 사용

Async/await

await 은 실행이 [오른쪽에서 왼쪽으로 실행]

함수가 Async 가 붙어야 사용 가능하다.

사용확인

const promise = new Promise(...)

promise.then((result) =>...)

const result = await promise; 사용 가능 .

- Async 도 promise를 간단하게 만든것.
- try{} catch(e) {} 문으로 실패를 구현한다.

AJax

서버로 요청을 보내는 코드

AJax요청을 axios 사용하면 편하다.

get요청 : 프로미스 기반 코드 async/ await 사용가능