2024.04.18

Node.js 비동기

논 블로킹 I/O ⇒ 한명이 일을 하는데 요리를 순차적으로 하지 않고 중간중간에 비는 시간이 있으면 다른 요리를 하는 것.
비동기 : 동시에 처리하지 않는다.

[라면 끓이는 순서]

1. 물을 끓인다. : 5분
2. 스프를 넣는다 : 10초
3. 면을 넣는다 : 10초
4. 라면을 더 끓인다: 3분
5. 파를 넣는다 : 10초
6. 먹는다. : 완성

- 현식(위 순서를 순차적으로 해내야 하기 때문에 8분 30초가 걸림
- node.js는 바로바로 넘어가기 때문에 5분이 걸림.

비동기 처리하는 방법

- 비동기 발생
  코드가 기다려야 하는 시간이 생긴다는 의미.
  순서를 무시하고, 일단 이전 시간이 오래 걸리면 다음 코드를 무작정 실행함.
  ex. setTimeOut(), setIntervel(), query

- 비동기 처리
  순서를 맞춰서 코드를 실행해주겠다.

이전 코드들의 시간을 다 기다려줘요. 비동기가 필요없을때가 있어요.(쿼리 기다려야할 때)
1. 콜백 함수 : 할 일 다하고 이거 실행해줘(= 순서 맞춰서 이걸 뒤에 실행해달라고)
2. promise(resolve, reject)
3. then & catch :
4. async & await

```
// Promise "객체" : 약속을 지키는 친구

let promise = new Promise(function(resolve, reject){ // 매개변수로
    // executor : 이 친구가 지켜야 할 약속

    setTimeout(() => resolve("완료!"), 3000);
    // 일을 다 하면 무조건 콜백함수 resolve 또는 reject 둘 중 하나는 호출
    // 할 일을 성공적으로 하면 resolve(결과), 실패하면 reject(에러)
}); // 친구 소환

// promise의 기본 메서드 : promise가 일 다 하고(= 약속 다 지키고) 호출해
promise.then(
    function(results){
        console.log(results);
    },
    function(error){}
);
```

- promise-chaining
  : result가 핸들러 체인을 따라 전달된다.

```
let promise = new Promise(function(resolve, reject){

    setTimeout(() => resolve("완료!"), 3000);
}).then(
```

```
    function(results){
        console.log(results);
        return results + "!!!!";
    },
    function(error){}
).then(
    function(results){
        console.log(results);
        return results + "!!!!!";
    },
    function(error){}
).then(
    function(results){
        console.log(results);
    },
    function(error){}
)
```

- async

```
//ascync-await : Promise 객체를 좀 더 쉽고 편하게 사용하는 문법
//즉, 비동기 처리가 쉽다

//async 함수
// __ function f(){} : 일반함수
// async function f(){}: async 함수

async function f(){
    return 7;
    //async 함수는 무조건 Promise 객체를 반환
    // 만약 반환값이 Promise가 아니면, Promise.resolve()로 묶어서 반환
}
```

```
f().then(
    function(result){
        console.log("promise resolve : ", result);
    },
    function(error){
        console.log("promise reject : ", error);
    }
)
```

- await

```
//ascync-await : Promise 객체를 좀 더 쉽고 편하게 사용하는 문법
//즉, 비동기 처리가 쉽다
//await은 async 함수 안에서만 동작
// await이 promise 객체.then() 이 메소드를 좀 더 쉽게 사용할 수 있는 방법

//async의 두번째 기능
//promise 객체가 일이 끝날 때까지 기다릴 수 있는 공간을 제공한다.

async function f(){

    //promise 객체 한개당 => query 하나
    let promise1 =new Promise(function(resolve,reject){
        setTimeout(()=> resolve("첫번째 쿼리"), 3000);
    });
    // promise 객체가 일 다 할때까지 기다려줌
    let result1 = await promise1;
    console.log(result1);

    //promise 객체 한개당 => query 하나
    let promise2 =new Promise(function(resolve,reject){
        setTimeout(()=> resolve("두번째 쿼리 with" + result1), 300
    });
```

```
    let result2 = await promise2
    console.log(result2);

    //promise 객체 한개당 => query 하나
    let promise3 =new Promise(function(resolve,reject){
        setTimeout(()=> resolve("세번째 쿼리 with" + result2), 300
    });

    let result3 = await promise3;
    console.log(result3);
}
f();
```