10/6 강의자료

1교시

01. async & await 개념

▼ async await

Promise를 활용해서 비동기 코드를 간결하게 작성하는 문법입니다.

async를 통해 비동기코드 라는 것을 명시해주고 그 안에 await을 사용해서 비동기 처리가 성공했을 경우를 반환 해준다. async 함수는 Promise 객체를 리턴해야하며 Promise가 아닌경우 Promise.resolve()로 감싸집니다. 또 다른 키워드 await는 async 함수 안에서만 동작합니다.

자바스크립트는 await 키워드를 만나면 프라미스가 처리될 때까지 기다립니다. 결과는 그 이후 반환됩니다.

예제코드를 보고 이해해보겠습니다.

함수를 호출하고, 함수 본문이 실행되는 도중에 `(*)`로 표시한 줄에서 실행이 잠시 '중단'되었다가 프라미스가 처리되면 실행이 재개됩니다. 이때 프라미스 객체의 result값이 변수 result에 할당됩니다. 따라서 위 예시를 실행하면 1초 뒤에 '완료!'가 출력됩니다.

await는 말 그대로 프라미스가 처리될 때까지 함수 실행을 기다리게 만듭니다. 프라미스가 처리되면 그 결과와 함께 실행이 재개되죠. 프라미스가 처리되길 기다리는 동안엔 엔진이 다른 일(다른 스크립트를 실행, 이벤트 처리 등)을 할 수 있기 때문에, CPU 리소스가 낭비되지 않습니다.

await는 promise.then보다 좀 더 세련되게 프라미스의 result 값을 얻을 수 있도록 해주는 문법입니다. promise.then보다 가독성 좋고 쓰기도 쉽습니다.

async,await 문법은 try, catch 문을 통해 error처리를 해줄 수 있습니다.

▼ promise VS async

Promise.all을 통해 비동기 코드를 동시에 실행시킬 수 있습니다. 비동기 코드를 멀티 스레드로 사용하는 느낌이죠.

하지만 async는 promise를 간결하게 사용할 수는 있지만 await 키워드에서 prmise가 resolove된 이후 다음 코드로 넘어가기 때문에 비동기 코드들을 싱글 스레드로 사용하는 느낌입니다.

그렇다면 asyc를 병렬처리하기 위해서는 어떻게 해야 할까요?

예제 코드를 통해 알아보겠습니다.

두 차이점을 알고 모두 사용 하실 수 있다면 더 효율적인 코드를 작성할 수 있을 것 입니다.

02. 철인 3종 경기

▼ await 유무 확인하기

await을 사용했을 때와 사용하지 않았을 때 어떤 차이가 있을까요?

03. 유저 정보 유청하여 rendering 하기

▼ CSS 라이브러리 사용하기

html 태그를 보면 css라이브러리를 link해왔습니다. 1주차에 배웠던 CSS 3가지 방법중 외부 파일을 만들어서 link를 통해 가져왔었죠?

동일하게 남이 이미 만들어 놓은 CSS 설정을 가져올 수도 있습니다.

그럼 여기서 integrity란 속성은 어떤 것일까요?

integrity란 진실성, 도덕성 이라는 뜻으로 암호값을 넣으므로써 파일이 변경, 변조되어 악용되는 사례를 막는 것입니다.

integrity 속성은 link, script 코드에서 사용할 수 있으며 실습에 사용된 코드에는 해시값으로 암호화 해놓았습니다.

이러한 보안은 HTTPS를 통해 거의 예방이 가능합니다.

crossorigin는 링크가 CORS정책을 지원하기 위한 속성으로

CORS는 Cross Origin Resource Sharing의 약자로 한국어 직역은 교차 출처 리소스 공유라는 뜻입니다.

이것을 풀어서 해석하자면 다른 출처로부터 온 리소스를 공유한다 라는 뜻입니다.

CORS : https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

2교시

04. fetch 사용하여 API 호출하기 3

▼ JS의 역사

전체적인 흐름인 https://string.tistory.com/114 에서 확인하시길 바랍니다.

우리가 배운 비동기의 역사를 살펴보면

CallBack ⇒ Promises(2015/ES6) ⇒ async/await (2017/ES8)

비동기로 많이 사용하는 HTTP 통신같은 경우는

XMLHTTPRequest(1999년) ⇒ ajax(2004년에 나오긴 했지만 2005년 구글이 구글 맵 만들면서 사용하기 전까진 관심밖) ⇒ fetch(2015/ES6) ⇒ axios

axios : https://axios-http.com/kr/

▼ fetch VS Axios

Fetch와 Axios 둘 다 HTTP 요청을 처리하기 위한 자바스크립트의 라이브러리이지만 몇 가지 차이점이 존재합니다.

1. Fetch의 경우 자바스크립트에 내장되어 있기 때문에 별도의 import나 설치가 필요하지 않습니다. 하지만 Axios의 경우 간단하지만, 위의 사용법 설명처럼 설치 과정이 필요합니다.
2. Fetch는 일부 예전의 인터넷 익스플로러 버전에서 지원하지 않는 경우가 있어, Axios가 Fetch보다 브라우저 호환성이 뛰어납니다.
3. Fetch에서는 지원하지 않는 JSON 자동 변환, 응답 시간 초과 설정 기능 등을 Axios에서 지원해줍니다.

05. CoffeeMaker 만들기

▼ 코드 흐름

index.js ⇒ App.js

```
const run = () => {
    window.addEventListener("DOMContentLoaded", () => {
        App();
    });
};

run();
```

App.js에서 우리가만들 CoffeeMaker.js를 실행시키고 리턴값을 사용합니다.

```
const coffeeMaker = CoffeeMaker(200, 2, render);

function render(maker) {
    const state = maker.getState();
    const { beans, beanPowder, coffee, currentMachine } = state;

    t(currentMachineText, `현재 남은 머신 대수 : ${currentMachine}`);
    t(coffeeBeanText, `남은 커피 콩 수 : ${beans}`);
    t(coffeePowderText, `남은 커피 파우더 : ${beanPowder}`);
    t(coffeeText, `남은 커피 : ${coffee}`);

    return Promise.resolve(maker);
}

addBeanButton.addEventListener("click", () => {
    const amount = Number(coffeeBeanInput.value);
```

```
    coffeeMaker.addBean(amount);
    coffeeBeanInput.value = "";
});

makeCoffeeButton.addEventListener("click", () => {
    // coffeeMaker를 구현 후에 잘 제조가 되는지 확인해보세요.
    coffeeMaker
        .prepareMachine()
        .then(render)
        .then((maker) => maker.grindBean())
        .then(render)
        .then((maker) => maker.brewPowder())
        .then(render)
        .then((maker) => maker.getCoffee())
        .then((coffee) => {
            t(statusText, `커피 ${coffee}ml 제조가 완료되었습니다.`);
            render(coffeeMaker);
        })
        .catch((error) => {
            t(statusText, error.message);
            render(coffeeMaker);
        });
});

checkStateButton.addEventListener("click", () => {
    const state = coffeeMaker.getState();
    t(document.getElementById("status"), JSON.stringify(state, null, 2));
});

render(coffeeMaker);
```

CoffeeMaker.js에서는 api.js과 store에서 만들어 놓은 함수와, constants.js에서 만들어 놓은 상수를 사용하여 작성합니다.

```
import { COFFEE_PER_BREW, BEANS_PER_BREW } from "./constants";
import { createAction, update, initializeState } from "./store";
import * as API from "./api";
```

store.js에서는 constants.js에서 만들어 놓은 상수를 사용하여 작성합니다.

```
import { BEANS_PER_BREW, COFFEE_PER_BREW } from "./constants";
```