Asynchronous

자료	Javascrtip
구분	Javascript
과목	

오늘 학습 내용이자. PDF요약은 다음과 같다.

- Asynchronous 비동기란 무엇인가?
- AJAX가 무엇인가?
- 이벤트루프에 대한 이해
- AXIOS가 무엇인가?
- Axios를 이용해 비동기처리방식 설명
    - callback 함수 방식
    - promise 방식
    - async/await 방식

1. 비동기란??

먼저, “동기” 라는 뜻은
프로그램의 실행 순서가 소스코드가 작성 된 순서대로 순차적으로 실행 되는 방식을 말한다.

비동기란? 소스코드가 작성 된 순서 대로 프로그램이 실행되는 도중
앞선 코드가 우선적으로 실행이 되었더라도 실행이 완료되는데 까지 시간이 걸린다면,

바로 다음에 실행 될 코드는 앞서서 실행한 코드가 완료될 때까지 기다리지 않고
즉시 실행되는 방식을 말한다.

즉, 특정 작업의 완료를 기다리지 않고 다음 작업을 실행하는 방식으로
프로그램의 실행 흐름이 순차적이지 않는 방식을 말한다.

예를 들어 카페에서 주문을 받는데 밀크쉐이크 주문이 들어 온 후에
아메리카노 주문이 들어왔다고 가정하자.

밀크쉐이크를 만들기 위한 믹서기가 돌아갈 동안에
아메리카노를 빠르게 만들어서 먼저 음료를 내어 주고 밀크쉐이크가 완성되면
바로 쉐이크를 내어 주면 된다. 따라서 후 순위의 작업이라도 먼저 처리 할 수 있다면

선순위 작업이 다 끝날 때 까지 기다릴 필요가 없는 것이다.
위의 예제가 비동기적으로 일을 처리한 케이스다.

그러나 카페 사장님이 융통성이 없어서 동기적으로 일을 처리 한다면 어떻게 될까?
밀크쉐이크가 다 완성될 때 까지 기다렸다가
밀크쉐이크가 완성이 되고 진동벨을 울려 손님에게 쉐이크를 넘겨 준 후에
그제서야 비로소 아메리카노 만들기를 시작 했었을 것이다.

그렇다면 이렇게 생각 해 볼 수도 있다.
그럼 모든 코드가 비동기적으로 돌아간다면 아주 효율적인 프로그래밍이 될 것 같은데 ?
굳이 동기적으로 코드를 실행 시킬 필요가 없지 않을까?

아니다. 반드시 동기적으로 일을 처리해야 하는 경우도 있다.
은행 업무를 예를 들어 보겠다.
은행에서 서버로
1. 계좌 전액 송금 요청이 들어온 후에
2. 입금 요청이 들어 왔다고 가정하자.

“송금” 프로세스는 “입금” 프로세스보다 더 복잡하다.
송금을 하기 위해서는 내 계좌 잔액을 확인하고 상대 계좌번호도 유효한지 확인한 후에 송금이
가능해 입금 프로세스 보다는 작업 처리 속도가 더 많이 소요된다.

지금 상황은 계좌 전액 송금을 한 후에 입금 처리를 해야 한다. 그런데 후순위 요청으로 들어온 입금
처리가 비동기적으로 우선 처리가 된다면?
송금 후 입금 되야 할 돈 까지 전부 다른 계좌로 이체가 될 것이다.
따라서 상황에 따라서 작업을 동기적으로 또는 비동기적으로 처리 할 상황이 각각
생길 수 있다는 것을 알아두자.

오늘은 JavaScript 를 이용해서 소스코드가 비동기 적으로 실행되게 하는것을 학습한다.
그런데 사실, 문제점이 하나가 있다.
JavaScript 언어는 여러가지 코드를 한번에 “병렬적”으로 실행시킬 수 없는 프로그래밍 언어 이다.
즉, javascript는 한번에 한가지의 일만 처리 할 수 있는 Single Thread 언어로
동시에 여러 작업을 처리할 수 없다.
(C++ Java Python은 여러 작업을 동시에 병렬로 처리할 수 있는 언어다)

그럼 JavaScript 로 어떻게 비동기 프로그래밍이 가능한 것일까?

1. 브라우저 환경에서는 JavaScript가 비동기 작업을 할 수 있도록 다양한 Web API를 제공해 준다.
   예를들어, “setTimeout” 함수 그리고 “AJAX api” 를 사용해서 JavaScript 코드가
   비동기적으로 작업을 처리 할 수 있게 도와준다.
   (setTimeout 그리고 AJAX api가 무엇인지는 곧 뒤에 학습할 예정이다.)

2. Node.js 환경에서는 c++로 개발된 libuv 라이브러리가 Javascript 언어의 비동기 처리 작업을
   도와준다.
   Node.js가 무엇일까? 앞서서 PDF에 한번 언급하기는 했다.
   처음에 JavaScript는 브라우저의 동적인 영역을 담당하기 위해서 만들어 졌지만,
   JavaScript가 점점 발전함에 따라 브라우저 밖에서도
   JavaScript로 코드를 실행시킬 수 있게 되었다.

   브라우저 밖에서 JavaScript를 실행시켜 주는 개발환경 (런타임)을 Node.js라고 하며,
   Node.js를 이용하면 JavaScript 를 가지고 서버도 구축 할 수 있다고 했다.

   여담으로 서버를 구축을 위한 대표적인 프레임워크로는

   - Spring Boot - Java 언어 기반 프레임워크로써 은행, 관공서, 공공기관프로젝트을 하는 대기업에서 많이 사용 한다. 장점은 높은 안정성과 보안성 그리고 유지보수에 용이한 구조라서 복잡한 시스템 또는 대형 프로젝트에 최적화 되어 있다.
   - Django - 파이썬 기반의 프레임워크이다. Instagram에서 사용 중이며, 빠른 개발 속도 그리고 ORM을 이용한 높은 트래픽과 데이터 처리에 특화되어 있다. 파이썬을 기반으로 하는 프레임워크다 보니, AI 모델 연동이 아주 쉽다는 장점이 있다.
   - Node.js - 아마존, 넷플릿스, 쿠팡, 우버, 카카오 에서 사용하며, 실시간 알림, 채팅, 빠른 사용자 응답이 필요한 서비스에 최적화단 프레임워크다. JavaScript 를 기반으로 하기 때문에 프론트엔드와 백엔드를 하나의 언어(JS)로 개발할 수 있다는 점이 큰 장점이다.

다시 본론으로 들어가,
브라우저 환경에서 Single Thread 언어인 Javascript가 비동기 처리가 가능하게 도와주는 도구에는
“AJAX api”가 있다고 언급했다. 그렇다면 AJAX가 무엇인지 알아보자.

AJAX가 무엇이냐?

AJAX 란? 웹페이지를 동적으로 만들기 위한 개발 기법을 말한다.

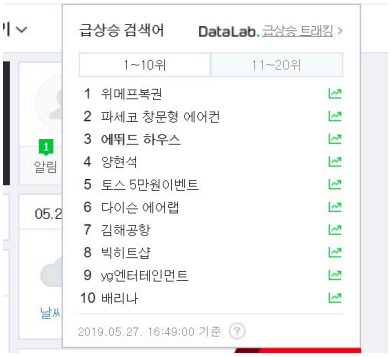
AJAX는 비동기를 구현하는 개발방식을 말한다.

AJAX는 (Asynchronous JavaScript And XML) 자바스크립트를 이용해 서버-클라이언트 간에 데이터를 주고 받는 HTTP 통신을 하는데

비동기적으로 통신할때 사용하는 개발기법이다. (언어 X 프레임워크 X)

프로그래밍 언어가 아니다.
프레임워크도 아니다. 그냥 개발 기법을 AJAX 라고 한다.

AJAX는 2005년 Google Maps 개발 시 사용되면서 전 세계적으로 유명해졌다.
AJAX는 비동기로 HTTP 통신을 할 때 MS에서 개발한 XMLHttpRequest(XHR) 객체를 이용한다.
그런데 XHR 객체를 이용하면 URL 통해서 서버 클라이언트 통신을 할 때
전체 페이지가 아닌 필요한 데이터만 별도로 새로고침! 없이! 불러올 수 있다.



예를들면 네이버 실시간 검색순위가 되겠다.
처음 실시간 검색순위가 나왔을 때만 해도 5분에 한번씩 웹페이지가 새로 고침이 되면서
전체 페이지를 다시 불러 왔었다.

하지만 AJAX 기술 도입 후
네이버 실시간 검색 순위가 바뀌더라도 네이버 페이지 전체가 다시 로드 되지 않았었다.
즉, 새로고침을 통해서 전체 페이지를 다시 불러오는 대신,
필요한 부분적 데이터만 업데이트 해서 사용자 경험을 향상 시켰다.

따라서 **AJAX는 XHR객체를 이용해서 새고로침 없이 필요한 데이터만 불러오는**
Javascript 개발기법 이라고도 할 수 있다.

이 AJAX개발 기법을 편리하게 사용할 수 있는 대표적인 JavaScript 라이브러리로는 “Jquery”가 있다.
그런데 Jquery는 Dom 조작하는 것이 어렵고 재사용성이 좋지 않아서

구글직원이 개발한 Angular 가 그 다음 JavaScript를 기반으로 하는 프레임워크의 대세가 되었었다.

그러나 Angular는 규모가 커지는 프로젝트를 하면 속도가 너무 느리다는 등의 여러 단점이 나온 다음에 그 다음 대세가 된 Javascript 기반 프로임워크로는 페이스북에서 만든 React 이다.
그리고 React는 아직 전 세계 개발자가 많이 사용하고 있다.

물론 우리가 학습할 Vue.js도 필요에 의해서 자주 사용되는 프레임워크 이다.
보통 규모가 있는 웹서비스의 전체 시스템은, 다양한 기술스택으로 구성되어 있는데
Vue.js는 빠른 프로토타입 제작 그리고 사용자경험을(UX) 중시하는 프로젝트에서 부분적으로 사용이 된다.

예를들어 네이버 웹툰 서비스에서는 Vue.js를 사용해서 사용자 인터페이스를(UI) 구현하고
네이버 쇼핑에서는 상품 리스트와 필터링 기능에서 Vue.js를 사용 했다.
카카오톡 웹 버전 그리고 카카오페이 결제과정에서 Vue.js를 이용해서 사용자 인터페이스를 구현했고
쿠팡이츠, 배민에서도 음식주문 내역, 음식 검색, 장바구니 기능 등
실시간으로 상품 정보를 업데이트 하는 파트에서 Vue.js가 사용되고 있다.

- GPT로 검색한 Vue.js의 장점은 다음과 같다.

> Vue.js의 장점
>
> 1. 빠른 프로토타입 제작: 작은 프로젝트에서 빠르게 프로토타입을 만들 수 있어 유용
>
> 2. 부분적인 사용 가능: 전체 애플리케이션에서 Vue.js를 한 부분만 사용해도 문제없음
>
> 3. 사용자 경험 중시: UI/UX 구현에 적합, 실시간 데이터 처리가 강점
>
> 4. 학습 용이성: React나 Angular보다 더 직관적이고 배우기 쉬움

실시간 댓글, 좋아요 기능 그리고 사용자 추천 시스템을 구현할 때에는
빠른 반응성 그리고 다양한 변수로 인한 유연성이 필요하다.
Vue.js는 우리가 아직 학습을 시작하지 않았지만
컴포넌트 기반 아키텍처 그리고 빠른 반응성 때문에 웹서비스에서 사용자 경험이 중요한 특정 부분을 담당하기도 하고 빠른 프로토타입을 제작을 위해서도 사용된다는 점을 기억해 두자.

다시 본론으로 돌아와서 이번에는 JavaScript 코드가 웹 API와 함께 비동기 처리를 어떻게 수행하는지 이해해 보자.

[이벤트루프 관련 영상인데 10분 40초 부터 실행하면 된다.]

https://www.youtube.com/watch?v=8aGhZQkoFbQ

JavaScript는 싱글 스레드로 작동하며, 기본적으로 동기적으로 코드를 실행한다. 하지만, 웹 API를 사용하면 비동기적으로 동작할 수 있다.

순서는 다음과 같다.

1. JavaScript 코드 실행
2. 비동기 함수가 호출되면, 해당 작업이 웹 API로 위임됨
3. 웹 API가 작업을 처리하고 완료되면, 콜백 함수나 프로미스가 실행됨
4. 이벤트 루프가 콜백 함수나 프로미스를 실행한다.

웹 API가 비동기 코드를 어떻게 처리하는지 이해를 했다면 그 다음은
비동기 코드 구현 및 실습에 앞서 이번에는 Axios가 무엇인지 살펴보자.

Axios가 무엇이냐?

Axios는 AJAX의 기능을 쉽게 사용할 수 있도록 만들어진 라이브러리 이다.

Axios는 주로 비동기적으로 데이터를 가져오거나 서버에 데이터를 전송할 때 사용되는 라이브러리

로써 Javascript로 HTTP 통신을 할 때 많이 사용하는 라이브러리가 바로 Axios다.

- AJAX는 XHR 객체 기반으로 비동기 통신을 지원하는 기술을 말하며
  - (새로 고침 없이 서버와 비동기적으로 데이터를 주고받을 수 있게 해주는 기술)
  - (데이터를 주고받을 때, 페이지 전체를 새로고침하는 것이 아니라, 필요한 부분만 갱신하는 기술)
- Axios는 AJAX 기술을 쉽게 사용할 수 있도록 만들어진 JavaScript 라이브러리로써
  "HTTP 통신을 비동기적으로 처리할 수 있도록 도와주는 라이브러리" 라고 할 수 있다.

Javascript로 HTTP 통신을 할 때 사용하는 라이브러리가 Axios 말고도 Fetch도 있다.

둘의 차이점을 이야기 하자면

1. Axios 는 대부분의 최신 브라우저에서 지원을 한다.
   반면에 Fetch는 지원하지 않는 브라우저도 존재한다. (Explorer 지원 안함)

2. Axios는 Axios를 통해서 응답받은 객체가 JSON으로 자동으로 적용이 되기 때문에 데이터를 다루기가 쉬운 반면에
   Fetch는 .json() 메서드를 사용하여 응답 데이터를 JSON 형식으로 따로 파싱을 해야 한다.

3. Axios는 모듈을 설치 후 import를 해서 사용한다.

하지만 Fetch는 Javascript 내장 라이브러리라서 별도로 import를 할 필요가 없다.

4. Fetch에는 없는 기능이 있다. 예를들면 response timeout 처리 하는 기능은 Fetch에는 없다.

이러한 이유로 인해서 웹 프론트 프레임워크 다룰 때에는 Axios를 가장 많이 선호 한다.

드디어 비동기 코드를 실습 해보자.

먼저 Axios 를 사용하기 위해 공식 홈페이지에 들어 간 후 cdn 설치하자. 그리고
아래 있는 javascript 예시 코드를 한번 작성해 보자.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
  <button>고양이사진이 나와라</button>

  <script>
    const URL = 'https://api.thecatapi.com/v1/images/search'
    const btn = document.querySelector('button')
    console.log('고양이가 어떻게 울지?')

    const cat = () => {
      axios({ method: 'get', url: URL, })
        .then((response) => {
          console.log(response)
          img_elem = document.createElement('img')
          img_url = response.data[0].url
          img_elem.setAttribute('src', img_url)
          img_elem.setAttribute('height', 100)
          document.body.appendChild(img_elem)
        })
        .catch(error => {
          console.log('이미지를 불러오는데 실패하였습니다.')
        })
    }

    console.log('고양이 울음소리를 아세요?')
```

```
    btn.addEventListener('click', cat)
    console.log('고양이는 냐옹냐옹 울지')

  </script>
</body>

</html>
```

라이브 서버를 켜고 F12를 눌러서 개발자 도구에서 console 창을 확인해 보자.

```
고양이가 어떻게 울지?
고양이 울음소리를 아세요?
고양이는 냐옹냐옹 울지
```

가 출력이 될 것이고,

"고양이사진이 나와라" 버튼을 클릭을 하면 사진이 나올 것이다.

하지만 위에서 코드가 작성한 순서를 보면

```
console.log('고양이가 어떻게 울지?')
console.log('고양이 울음소리를 아세요?')
btn.addEventListener('click', cat)
console.log('고양이는 냐옹냐옹 울지')
```

위와 같이 되어 있을 것이다.

동기식으로 코드가 진행이 되었다면

3번 째 줄 click 을 해서 고양이 사진이 나온 다음에

console.log('고양이는 냐옹냐옹 울지') 부분이 실행이 되어야 할 것이다.

동기식 코드는 작업이 시작하는 순서대로 출력이 되지만

비동기식 코드는 작업이 완성되는 순서대로 출력이 된다고 했다.

그렇다 보니, 버튼을 눌러 서버로부터 고양이 사진을 받아오기 전에

'고양이는 냐옹냐옹 울지' 가 먼저 출력이 되는 것이다.

이와 같이 비동기적으로 소스코드를 작성을 한 후에 실행을 해 보면

서버에서 먼저 응답을 해 주는 데이터부터 우선적으로 화면에 출력이 된다.

그렇다 보니, 개발자 입장에서는 소스코드의 실행 순서가 불 분명 해진다는 단점이 있다.

(소스코드 실행 순서가 서버의 응답속도에 의존하게 되므로...)

그래서 개발자는 비동기 코드에도 순서를 정해줘서 비동기 데이터의 처리 순서를 통제해야 한다.
서버에서 응답 받은 순서대로 화면에 출력 되는 것이 아니라,
개발자의 의도대로 코드가 실행 되게 해야 할 것이다.

자바스크립트에서 **Axios를 사용한 HTTP 통신에서 비동기 데이터 처리하는 방식**은 여러가지가 있다.
그 중 대표적인 3가지 방식을 비교해 보자.

1. 콜백함수 방식
2. Promise 방식
3. Async/Await 방식

---

1. 콜백함수 방식

비동기 방식은 요청과 응답의 순서를 보장하지 않기 때문에
서버의 응답의 처리 속도에 의존하는 경우에
콜백 함수를 이용하여 작업 순서를 간접적으로 끼워 맞출 수 있다.

예를 들어 num 이라는 변수에 숫자를 넣을 것인데
변수에 들어 갈 숫자를 서버로 부터 3초후에 응답을 받는다고 가정하자.

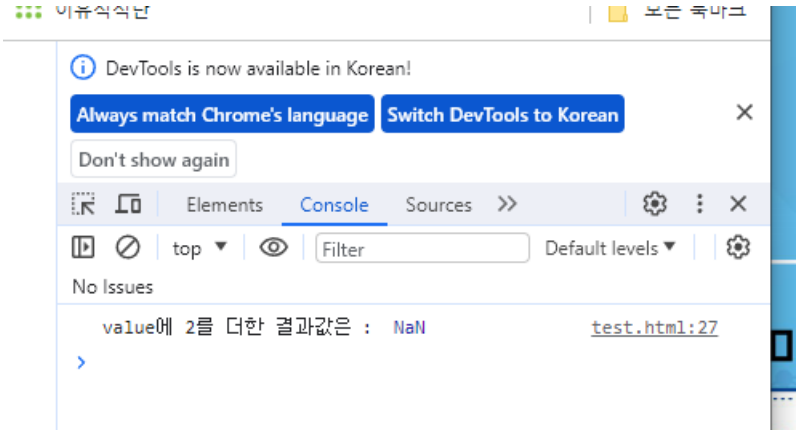
서버로 부터 변수 값을 응답 받은 후,
응답 받은 값에 2를 더한 값으로 바꾼 후 출력하는 코드를 작성 해 볼 것이다.

코드는 아래와 같을 것이다.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

  <script>
    function creatNum_database() {
```

```
    let value
    setTimeout(() => {
      value = 100

    }, 3000)

    return value
  }

  function calcul() {
    let num = creatNum_database();
    num += 2
    console.log('value에 2를 더한 결과값은 : ', num)

  }

  calcul();

 </script>
</body>

</html>
```

개발자 도구를 열어서 console 창을 확인해 보라

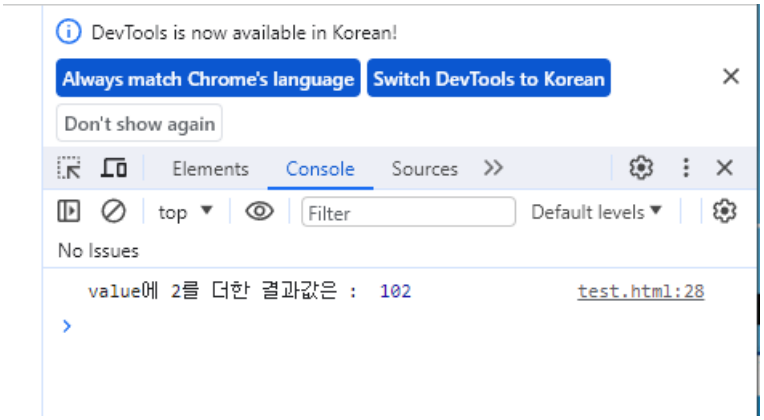


Not a None 이 출력이 되었다. 이는 서버로 부터 데이터를 응답받기 전에
결과값이 먼저 출력이 되어서 NaN이 출력이 된 것이다.

그러면 이번에는 우리의 의도대로 서버로부터 정수값을 응답받은 후

2를 더해 출력 할 수 있도록 코드를 수정해 보겠다.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

  <script>

    function creatNum_database(callback){
      setTimeout(()=>{
        let value=100;
        callback(value);

      },3000)
    }

    function calcul(){
      creatNum_database(function(value){
        let num=value+2;
        console.log('value에 2를 더한 결과값은 : ',num)
      })
    }

    calcul();

  </script>
</body>

</html>
```



위 코드의 프로세스가 한번에 이해가 되지 않을 수 있으니

과정을 상세하게 설명해 보자면 다음과 같다.

```
...
 <script>

  function creatNum_database(callback){
   setTimeout(()⇒{
     let value=100;       // 3. 3초후 100생성후,
     callback(value);     // callback함수의 인자값으로 100 들어감

   },3000)
  }

  function calcul(){
   creatNum_database(function(value){   // 2. creatNum_database함수 호출
     let num=value+2;      // 4. 100에 2를 더하고 출력
     console.log('value에 2를 더한 결과값은 : ',num)
   })
  }

  calcul(); // 1. calcul함수호출
 </script>
...
```

1. calcul 함수 호출
2. calcul 함수에서 database함수 호출하는데 인자값으로 callback함수 들어감
3. 3초후 value에 100이 생성되고 callback 함수로 100을 넣음
4. value에 2를 더한 후 출력

이러한 방식으로 callback 함수를 이용해서 비동기 작업의 순서를 억지로 껴 맞출 수 있다.

하지만, callback 함수로 비동기 코드의 작업순서를 맞추다 보면 단점이 발생한다.

만약에 작업이 여러개가 있어서 여러개의 작업 순서를 껴 맞춰야 하는 경우에는
콜백 함수 안에 콜백함수가 들어가고 또 다른 작업 순서를 맞추기 위해서 그 콜백함수 안에
또 다른 콜백함수가 들어가는 과정이 반복이 될 수도 있다. 즉, 콜백 지옥이 시작이 된다.
(함수안에 함수가 들어가고 또 그 함수안에 다른 함수고 또 들어가고...)

다시말해, 콜백지옥(callback hell) 의 형태의 코드는 가독성 뿐만 아니라 유지보수에도 좋지 않다.
뿐만아니라 만약에 서버로 부터 응답을 못 받을 경우에는
후순위에 있는 뒤에 있는 작업은 아무것도 실행이 안 될 것이다.
그래서 이러한 단점을 Javascript는 Promise 객체라는 것을 만들어 극복하였다.

---

2. Promise 방식
promise를 이용하면 비동기 작업을 깔끔하게 연결 시킬 수 있을 뿐더러
database로 부터 응답을 받지 못한 경우에도 대응할 수 있다.

Promise를 이용한 코드를 살펴보자.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

  <script>

    function creatNum_database() {

      return new Promise((callback) => {

        setTimeout(() => {
          let value = 100;
          callback(value);

        }, 3000)
```

```
    })

  }

  function calcul() {
    creatNum_database()            // 2. creatNum_database함수 호출
      .then(value => {
        let num = value + 2;
        console.log('value에 2를 더한 결과값은 : ', num)
      })
      .catch(error => {
        console.log(error);
      })
  }

  calcul();  // 1. calcul 함수 호출

</script>
</body>

</html>
```

1. calcul 함수가 호출된다.
2. calcul 함수 안에서 creatNum_database 함수가 호출되고 3초 후에 100을 반환하는 Promise를 생성한다.
3. Promise가 해결되면 .then()의 콜백함수가 실행되고 value+2 가 계산 후 출력 된다.
4. 만약에 응답에 오류가 발생하면 .catch()에서 오류를 처리한다.

그런데 Promise 코드를 살펴보니, .then 그리고 .catch가 보인다.
아까 맨 처음 Axios 라이브러리에 대해서 설명하면서 고양이 사진 뽑는 소스코드와 상당히 유사하다.

그 이유는 Axios가 promise 기반으로 만들어 진 (Promise API를 활용 한) HTTP통신 라이브러리
이기 때문이다.

따라서 Axios가 무엇이냐? 라고 묻는다면,
"Promise 기반으로 만들어 진 HTTP 비동기 통신 라이브러리 입니다. "
라고 하면 대답을 하면 정확한 답변이 되겠다.

추가적으로 Javascript에서 비동기 코드를 사용하기 위한 3번째 방법인
Async/await 방식의 코드도 한번 살펴보자.

3. async/await 방식

async 와 await 는 동기적으로 작성하는 소스코드와 생김새가 같아서

사용법도 간단하고 이해하기도 쉽다.

function 키워드 앞에 async 만 붙여주면 되고,

비동기로 처리되는 부분 앞에 await 만 붙여주면 된다.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

  <script>


    function creatNum_database() {
      return new Promise((callback) => {

        setTimeout(() => {
          let value = 100;
          callback(value);
        }, 3000)
      })
    }

    async function calcul() {

      try {
        let num = await creatNum_database()
        num += 2
        console.log('value에 2를 더한 결과값은 : ', num)
      }
      catch (error) {
        console.log(err)
      }

    }
    calcul();

  </script>
```

```
</body>

</html>
```

개발자 도구를 열어 다시 확인 해보자.

await는 항상 async 함수 안에서 실행되며, async는 항상 Promise 객체를 반환한다.

async/await 를 사용하면 await 키워드로 비동기 처리를 기다리고 있다는 것을 직관적으로

표현하고 있음을 볼 수 있다. 이는 코드의 가독성을 높여 유지보수를 용이하게 해준다는 장점이 있다.

[참고]

추가적으로 보너스 개념으로 fetch 도 하나 더 살펴보자.

다시, 위에서 언급했던 내용을 중간 정리 겸 리마인드를 하자면

Javascript에서 비동기HTTP 통신을 위한 라이브러리로 2가지가 있다고 말했었다.

Axios도 있고 Fetch도 있다고 했었다!

Fetch는 Axios와 다르게 javascript 내장 라이브러리를 사용하기 때문에 따로 cdn이 따로 필요가 없다.

그리고 Fetch 또한 Axios와 마찬가지로 Promise 기반으로 구성되어 있어 비동기 처리에 편리하다.

하지만 응답 데이터를 JSON 형식으로 따로 파싱을 해야 한다고 위에서 말했다.

Axios와 Fetch 차이점을 설명하면서 언급을 했던 것 같다.

기본 코드는 다음과 같다

```
async function logJSONData() {
  // url로 요청 보내기
  const response = await fetch("http://example.com/movies.json");

  // json 데이터로 변환
  const jsonData = await response.json();

  // json 데이터 출력
  console.log(jsonData);
}
```

https://developer.mozilla.org/ko/docs/Web/API/Fetch_API/Using_Fetch

공식문서에 들어가면 위 코드가 그대로 있을 것이다.

그럼 위에서 했던 axios를 사용해서 고양이 사진 뽑았던 것을
fetch 와 async/await 를 이용해서 구현해 보겠다. 구현은 간단하다.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
  <button>고양이사진이 나와라</button>

  <script>
    const URL = 'https://api.thecatapi.com/v1/images/search'
    const btn = document.querySelector('button')
    console.log('고양이가 어떻게 울지?')

    async function cat(){
      let img_url;
      try{
        response=await fetch(URL)
        data=await response.json()
        img_url = data[0].url
        img_elem = document.createElement('img')
        img_elem.setAttribute('src', img_url)
        img_elem.setAttribute('height', 100)
        document.body.appendChild(img_elem)
        console.log('고양이는 냐옹냐옹 울지')
      }catch(error){
        console.log('이미지를 불러오는데 실패하였습니다.')
      }
    }

    console.log('고양이 울음소리를 아세요?')
    btn.addEventListener('click', cat)

  </script>
</body>

</html>
```

함수 앞에 async 를 붙이면 안에서 await 을 쓸 수 있다.

fetch 함수의 인자로 URL주소를 넣고, fetch 의 결과는 Promise 객체를 반환한다.

이후 json() 함수 처리를 한 다음 img를 파싱하면 된다.

오늘 AJAX / Axios / Fetch 에 대해서 살펴 보았고

Javascript에서 비동기 HTTP 통신을 하면서 작업 순서를 컨트롤 하기 위한 방법으로

콜백함수 사용하는 방법 // Promise객체 사용하는 방법 // async-await 사용하는 방법을 살펴 보았다.

각 사용법을 살펴 보자면 아래와 같은 표로 정리를 할 수 있겠다.

	콜백함수	Promise	Fetch	async/await
프로미스 지원 여부	안함	함	함	함
성공 핸들링	콜백구현	.then()	.then()	try.. Catch문 사용
에러 핸들링	콜백구현	.catch()	.catch()	try.. Catch문 사용
가독성	별로	쏘쏘	굳	굳
사용빈도	별로	굳	굳	굳

우리 교안에서는 콜백함수 그리고 Pomise만 나왔다.

따라서 실습시 Promise만 잘 사용해도 무관하다.

하지만 async/await 정도는 알아두는 것을 추천한다.

<끝>