Asynchronous

자료	Javascrtip
구분	Javascript

Asynchronous 비동기란 무엇인가?

먼저 동기 라는 뜻은 프로그램 실행 흐름이 소스코드가 작성 된 순서대로 순차적으로 실행 되는 방식을 말하며

비동기란? 코드가 먼저 시작 되었더라도 실행이 완료되는데 시간이 걸린다면, 실행되고 있는 코드가

완료될 때까지 기다리지 않고 바로 다음 코드가 실행되는 방식을 말한다. 즉, 프로그램의 실행 흐름이

순차적이지 않는 것을 말한다.

예를 들어 카페 사장님이라면 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 음료를 내어 주면 된다. 따라서 후 순위의 잡업 순서라도 먼저 처리 할 수 있다면
선순위 작업이 다 끝날 때 까지 기다릴 필요가 없는 것이다. 즉, 비동기적으로 일을 처리한 케이스다.

그럼 카페 사장님이 융통성이 없어서 동기적으로 일을 처리 한다면 어떻게 될까?

밀크쉐이크가 다 완성될 때 까지 아메리카노 만들기를 시작하지 않고 기다렸다가,

밀크쉐이크가 다 완성이 되고 진동벨을 울려 손님에게 건내 준 후에 아메리카노 만들기를 시작 했었

을 것이다.

그럼 모든 코드가 비동기적으로 돌아간다면 아주 효율적인 프로그래밍이 될 것 같은데 동기적으로

코드를 실행 시킬 필요가 없지 않을까?

아니다. 반드시 동기적으로 일을 처리해야 하는 경우가 있다. 예를들어 은행에서 서버로

계좌 전액 송금 요청이 들어온 후에 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 따라 동기적으로 또는 비동기적으로 작업을 처리 할 상황이 생긴다.

자 그래서 오늘은 javascript 비동기 적으로 소스코드가 실행 될 수 있도록 소스코드를 작성하는 것을 학습 할 것이다.

그런데 문제가 생겼다. javascript 언어는 여러가지 코드를 한번에 병렬적으로 실행시킬 수 없는 언어다. 즉, javascript는 한번에 하나의 일만 처리 할 수 있는 single Thread 언어로 동시에 여러 작업을 처리할 수 없다.

그럼 javascript 어떻게 비동기 코드를 실행 시킬까?

1. 브라우저에서는 별도로 제공하는 api 예를들면 setTimeout 그리고 ajax api 를 사용해서
   javascript 코드가 비동기적으로 작업을 처리 할 수 있게 된다.

2. node에서는 c++로 개발된 libuv 라이브러리를 통해서 javascript 언어의 비동기처리 작업을 도와준다.

   node가 무엇? 브라우저 밖에서도 JavaScript로 코드를 실행시켜주는 개발환경(런타임)

을

말하하는데 node.js 프레임워크를 이용하면 javascript 를 가지고 서버도 구축 할 수 있다.

참고로 서버를 구축을 위한 대표적인 프레임워크로
spring - 관공서관련, 정부관련일 하는 대기업들 (java 언어를 기반으로 함)
node.js - 구글 아마존 ms 넷플릿스 ebey uber naver kakao coupang 에서 많이 사용을 하며

node를 공부할때 같이 하면 좋은 언어랑 그리고 프레임워크 으로는 typescript 그리고 nest.js 등이 있겠다.

다시 본론으로 들어가 브라우저에서 제공하는 ajax api를 이용하면 javascript 코드를 비동기적으로

실행 시 킬 수 있다고 하는데.. 그럼 ajax가 무엇인지 알아야 겠다.

ajax가 무엇이냐?

ajax 란? 웹페이지를 동적으로 만들기 위한 개발 기법 이라고 생각하면 된다.
ajax는 비동기를 구현하는 개발방식을 말한다.

ajax는 (Asynchronous JavaScript And XML) 자바스크립트를 이용해 서버 클라이언트 간에 데이터를 주고받는 HTTP 통신을 하는데
비동기적으로 통신할때 사용하는 개발기법이다. (언어 X 프레임워크 X)

2005년 구글맵스 개발시 사용되면서 전 세계적으로 유명해졌다.

Ajax는 비동기로 HTTP 통신을 할 때 MS에서 개발한 XMLHttpRequest(XHR) 객체를 이용하는데
이 객체를 이용하면 URL 통해서 서버 클라이언트 통신을 할 때
전체 페이지가 아닌 필요한 데이터만 별도로 새로고침 없이 불러올 수 있다.

예를들면 네이버 실시간 검색의 데이터가 바뀌더라도 네이버 페이지 전체가 다시 로드 되지 않았었다.

따라서 Ajax는 새로고침 없이 필요한 데이터만 불러오는 javascript 개발기법 이라고도 할 수 있다.

이 개발 기법을 편리하게 사용할 수 있는 대표적인 javascript 라이브러리로는 jquery가 있다.

그런데 jquery는 dom 조작하는 것이 어렵고 재사용성이 좋지 않아서 구글직원이 만든 angular 가 그 다음 javascript 라이브러리의 대세가 되었다.

그러나 angular는 규모가 커지는 프로젝트를 하면 속도가 너무 느리다는 등의 여러 단점나온 다음에
그 다음 대세가 된 javascript 라이브러리는 페이스북에서 만든 react다. 그리고 react는 아직도 많이 사용하고 있다.

다시 본론으로 돌아와서

javascript 코드가 웹API와 함께 어떤 과정을 통해서 비동기처리를 하는지 미리 살펴 보자.

간단하게 한번 더 보고 싶다면 라이브 수업을 봐도 좋고 아래 영상을 참고하라

[이벤트루프 관련 영상]

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

11분 50초 부터 실행하면 된다.

자 오늘 우리는 javascript를 통해서 자바스크립트로 HTTP 통신을 할 것이다.

javascript로 http 통신을 할 때 많이 사용하는 라이브러리가 Axios다.

Axios는 브라우저 뿐만 아니라 node.js를 위한 http비동기 통신 라이브러리다.

javascript로 http 통신을 할 때 사용하는 라이브러리가 Axios 말고도 fetch가 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 json을 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 써야 하지만 fetch는 javascript 내장 라이브러리라서 별도로 import를 할 필요가 없다.
4. fetch에는 없는 기능이 있다. 예를들면 response timeout 처리 하는 기능은 fetch에는 없다.

이러한 이유들로 인해서 웹 프론트 프레임워크 다룰 때에는 axios를 많이 선호 한다.

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>
```

동기식 코드가 다 완성이 되고 비동기 코드인 사진띄우기가 완성이 되었다.

```
고양이는 어떻게 울지?

고양이 울음 소리를 아세요? 이후에

버튼을 클릭해서 고양이 사진이 나온 다음에

고양이는 냐옹냐옹 울지 를 출력 하고 싶다.
```

동기식 코드는 작업이 시작하는 순서대로 출력이 되지만

비동기식 코드는 작업이 완.성.되.는. 순서대로 출력이 된다.

그렇다 보니, 버튼을 눌러 서버로 부터 고양이 사진을 받아오기 전에

'고양이는 냐옹냐옹 울지' 가 먼저 출력이 되버린다. (개발자 도구 console 창으로 확인하자)

이와 같이 비동기적으로 소스코드를 작성을 한 후에 실행을 해 보면 서버에서 응답을 먼저 해 주는
데이터 부터 먼저 출력이 된다. 그렇다 보니, 개발자 입장에서는 소스코드의 실행 순서가 불분명 해진다는 단점이 있다.

그래서 비동기 코드에도 순서를 정해줘서 비동기 데이터의 처리 순서를 통제해야 한다.
서버에서 응답받은 순서대로 출력되는 것이 아니라, 개발자의 의도대로 코드가 실행 되게 해야 할 것이다.

자바스크립트에서 비동기 데이터를 처리하는 방식은 여러가지가 있다. 그 중

1. 콜백함수 방식
2. Promise 방식
3. async/await 방식 을 살펴 볼 것이다.

---

1. 콜백함수 방식

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

예를 들어 num 이라는 변수에 숫자를 넣을 것인데

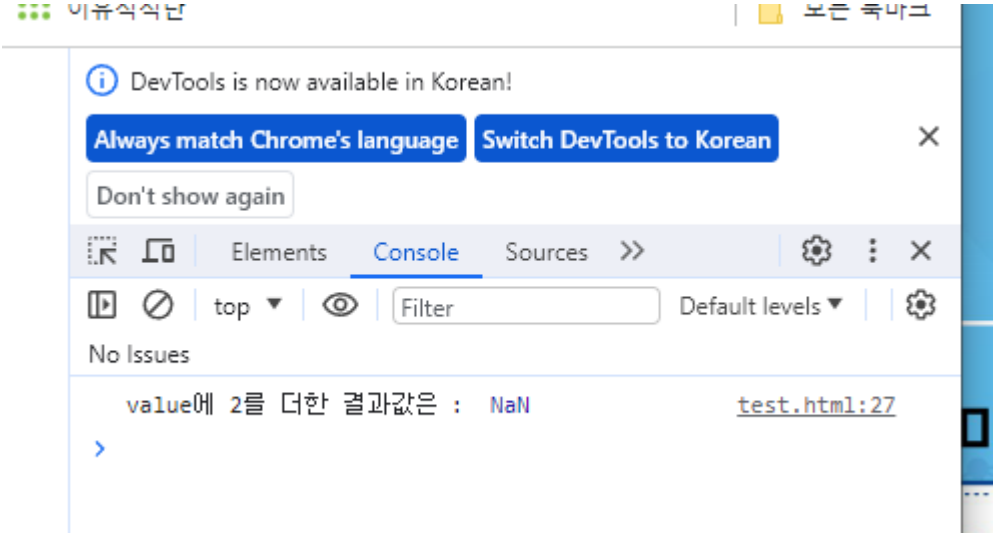
변수에 들어 갈 숫자를 데이터베이스로 부터 3초후에 응답을 받는다고 가정하자.

데이터 베이스로 부터 변수 값을 응답 받은 후 2를 더한 값을 출력하는 코드가 있다고 하자.

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

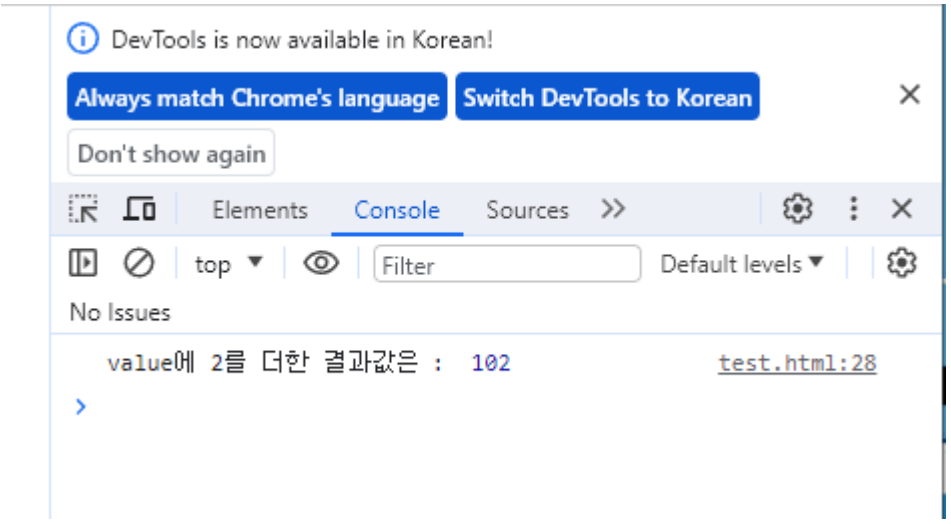


Not a none 이 출력이 되었다. 이는 database로 부터 숫자를 응답받기 전에 결과값을이 먼저 출력이 되어서 NaN이 출력이 되었다.

그러면 이번에는 의도대로 데이터베이스로 부터 정수값을 응답받은 후 2를 더할 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 와닫지 않을 수 있으니 과정을 상세하게 설명해 보자면 다음과 같다.

```
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
```

이런식으로 callback 함수를 이용해서 비동기 작업의 순서를 억지로 껴 맞출 수 있다.
하지만, callback 함수로 비동기 코드의 작업순서를 맞추다 보면 단점이 발생한다. 만약에 작업순서가 여러개인 경우에는 콜백 함수 안에 콜백함수가 들어가고 또 실행 순서를 맞추기 위해서서 그 콜백함수 안에 또다른 콜백이 들어가는 과정이 반복이 될 수도 있다. 즉, 콜백 지옥이 시작이 된다.

다시말해, callback hell 이 발생할 수 있을 뿐더러 가독성이 좋지 않다.
뿐만아니라 만약에 database로 부터 응답을 못 받을 경우에는 뒤에 있는 작업을 아무것도 실행이 안 될 것이다.

그래서 이러한 단점을 javascript는 Promise 객체라는 것을 만들어 극복하였다.

Promise를 이용하면 비동기 작업을 깔끔하게 연결시킬 수 있을 뿐더라
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()
        .then(value => {
          let num = value + 2;
          console.log('value에 2를 더한 결과값은 : ', num)
        })
        .catch(error => {
          console.log(error);
        })
    }

    calcul();
```

```
  </script>
</body>

</html>
```

그런데 promise 코드를 보니 이 문서에서 맨 처음 axios 에 대해서 설명하면서 고양이 사진 뽑는 코드를 작성 했었는데 그 코드와 상당히 유사하다.

그 이유는 axios가 promise 기반으로 만들어 진 Promise API를 활용 한 HTTP통신 라이브러리 이기 때문이다. 따라서 axios가 무엇이냐 라고 뭍는다면, promise 기반으로 만들어 진 HTTP 비동기 통신 라이브러리 라고 하면 정확한 정의가 되겠다.

추가적으로 javascript에서 비동기 코드를 사용하기 위한 3번째 방법인 aync/await 방식의 코드도 한번 살펴보자.

async 와 await 는 동기적으로 작성하는 코드와 같아 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 확인하자.

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

따라서 원한다면 .then 사용도 가능하다. 뿐만 아니라. 에러처리까지 하려면 위에서 예를 들었던 코드와 같이 try/catch 문을 사용하여 에러를 처리를 할 수 있다.

promise는 then 메서드를 연속적으로 사용하여 비동기 처리를 하지만,

async/await는 await 키워드로 비동기 처리를 기다리고 있다는 것을 직관적으로 표현하고 있음을 볼 수 있다.

async/await의 장점은 비동기적 접근방식을 동기적으로 작성할 수 있게 해주어 코드가 간결해지며

가독성을 높여져 유지보수를 용이하게 해준다.

추가적으로 하나만 더 살펴보자. fetch 도 살펴보자.

다시 리마인드 하자면

javascript에서 비동기HTTP 통신을 위한 라이브러리로 2가지가 있다고 말했었다.

axios도 있고 fetch도 있다고 했었다!

fetch는 axios와 다르게 javascript 내장 라이브러리를 사용하기 때문에 따로 cdn이 필요가 없다.

그리고 fetch 또한 axios와 마찬가지로 Promise 기반으로 구성되어 있어 비동기 처리에 편리하다.

하지만 응답 데이터를 JSON 형식으로 따로 파싱을 해야 한다고 axios와 fetch 차이점을 설명하며 앞서 언급했다.

기본 코드는 다음과 같다

```
async function logJSONData() {
  const response = await fetch("http://example.com/movies.json"); // url로 요청 보내기
  const jsonData = await response.json();                    // json 데이터로 변환
  console.log(jsonData);                                  // json 데이터 출력
}
```

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 프로미스 객체를 반환한다.
이후 json() 함수 처리 후 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. <끝>

	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