

05



Ajax를 사용하기 위한 라이브러리

Web APIs

# Ajax

프론트의 js 코드 내에서 백엔드 서버와 네트워크 통신을 하기 위한 프로그램

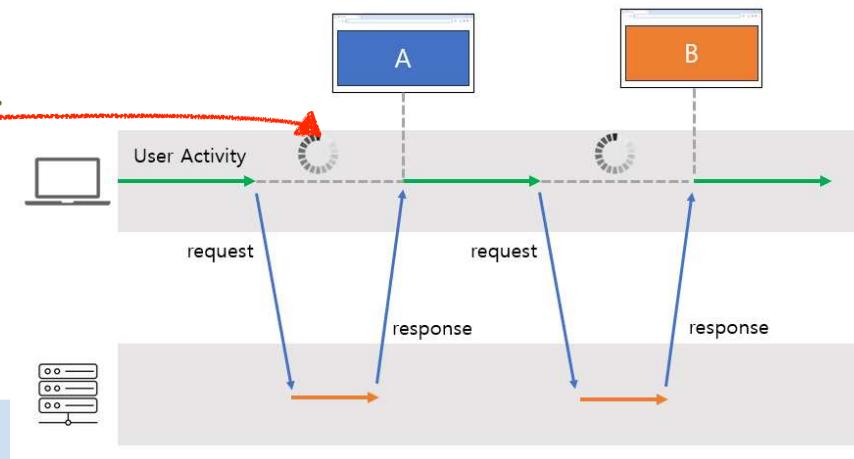
요즘 서버 연동은 거의 Ajax로 하기에 필수다.

- Ajax는 서버 연동을 위한 자바스크립트 프로그램입니다.
- Ajax는 ~~Asynchronous Javascript XML~~의 약어이며 ~~비동기 서버 연동~~을 위해 제공되는 기술입니다.

네트워킹은 시간이 오래 걸린다. 서버에 요청한 정보가 넘어오기까지 대기하지 않는다.

- 브라우저에서 서버에 네트워크 통신을 할 때 Ajax를 제외한 모든 통신은 동기 통신입니다. (동기 통신이 웹의 기본이다)
- 화면에 링크가 있고 그 링크를 사용자가 클릭해서 서버와 통신하는 경우, 화면에 버튼이 있고 그 버튼을 클릭해 서버와 통신하는 경우 등 대부분의 통신은 동기적 통신입니다.
- 동기 통신이란 브라우저에서 서버에 요청(request)을 보내면 서버에서 응답(response) 할 때까지 브라우저는 대기상태가 된다는 의미입니다.

단순히 다른 코드가 실행되지 않는다는 게 아니라  
브라우저 자체가 대기상태가 되어서 아무것도 안 된다.  
(입력, 클릭 등의 동작 불가)



# Ajax

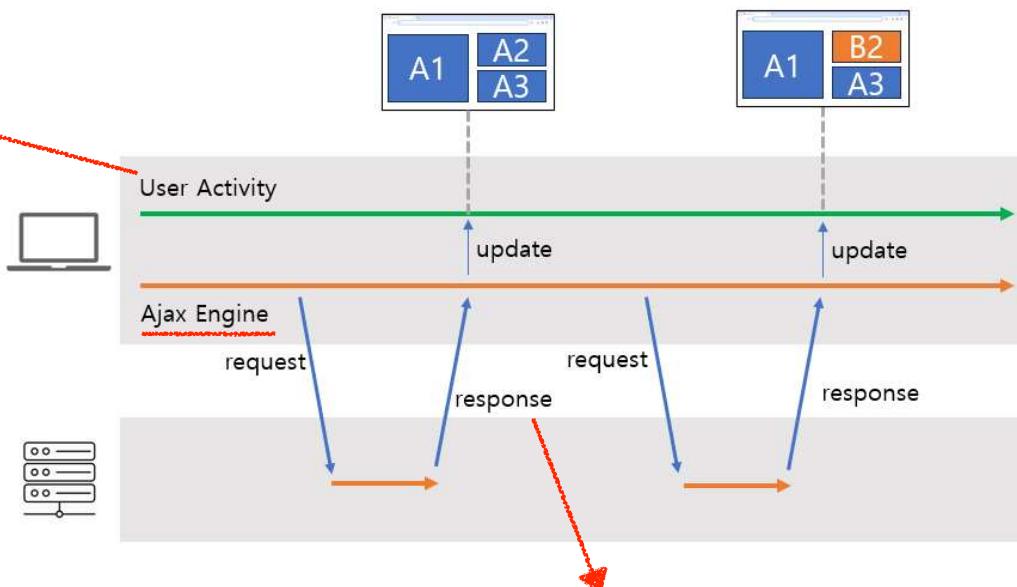
---

- 서버에 요청을 보내게 되면 브라우저는 이 요청을 동기적으로 처리하게 됨으로 응답이 올때까지 대기하게 되며 이로 인해 사용자는 브라우저를 이용할 수 없게 됩니다.
- 동기 통신을 하면 서버의 응답으로 전체 화면이 갱신되게 됩니다.
- 동기 통신의 단점을 해결하고자 나온 기술이 비동기 통신을 지원하기 위한 Ajax 입니다.
- 비동기 통신은 브라우저에서 서버에 요청을 동기적으로 처리하지 않는 통신입니다.
- 즉 서버에 요청을 보내고 응답이 올때까지 대기하지 않는다는 의미입니다.
- 비동기 통신을 이용하게 되면 브라우저가 서버와 통신을 한다고 하더라도 브라우저는 대기상태가 되지 않음으로 사용자는 지속적으로 브라우저를 이용할 수 있게 됩니다.
- 또한 서버에서 넘어온 결과로 사용자의 화면을 전체 갱신하지 않음으로서 서버에서 넘어온 결과로 사용자가 보는 화면의 일부분만 갱신하는 것이 가능해 집니다.
-

# Ajax

- 브라우저의 서버 연동은 기본이 동기통신인데 이 비동기 통신이 가능한 이유는 브라우저에 있는 Ajax Engine에 의해 Ajax 통신에 한해서만 비동기적으로 처리되기 때문입니다.

유저는 lock 걸리지 않는다.



새롭게 받은 부분만 렌더링된다.

# XMLHttpRequest

- Ajax 통신으로 서버에 요청을 보내려면 XMLHttpRequest 객체를 이용해야 합니다. 전체적인 작업의 흐름은 다음과 같습니다.

1. XMLHttpRequest 객체 생성 **new 연산자로 생성**
2. 요청 준비(초기화) **정보를 주는 단계**
3. 요청 전송
4. 이벤트 콜백을 통한 결과 이용

**안 기다릴 거니까, 비동기 이벤트 콜백 함수 이용**

# XMLHttpRequest

객체

- XMLHttpRequest의 메서드는 아래의 것들이 있습니다.

- open() : 요청 초기화
- abort() : 요청 취소
- send() : 서버 요청
- setRequestHeader() : 요청 헤더 설정

필수

[ 1장 overview에서 다룬 내용 ]

HTTP 규약에 의해 서버로 보내는 데이터,  
서버에서 보내는 데이터의 형식은 표준이 있다.  
데이터의 형식은 header와 body로 나눌 수 있다.

# XMLHttpRequest

요청을 위한 정보를 담는 행위

- XMLHttpRequest 객체를 생성한 후 open() 함수로 새로운 요청을 위한 초기화 작업을 해야 합니다.
- open() 함수에 요청을 위한 정보가 담겨야 하는데 첫번째 매개변수가 HTTP Request Method 입니다.
- GET, POST, PUT, DELETE 등의 요청 방식을 정의하고 두번째 매개변수에 요청 URL 을 지정합니다.
- 그리고 세번째 매개변수에 서버 요청을 동기 통신을 할 것인지, 비동기 통신을 할 것인지를 true 혹은 false 값으로 지정합니다. true 는 비동기 통신이며 false 는 동기 통신입니다. false 로 지정하게 되면 서버에서 응답이 올 때 까지 대기하게 됩니다.
- 초기화된 요청을 서버에 전송하는 함수는 send() 함수입니다.

Ajax 요청 전송

```
1 let xhr = new XMLHttpRequest();
2 xhr.open('get', `http://localhost:3000/sum/${numNode.value}`, true);
3 xhr.send();
```

여기서 받은 데이터를 이벤트 콜백 함수의 매개변수로 활용할 수 있다.  
(다음 페이지)

# XMLHttpRequest

---

- send() 함수로 서버에 보낸 요청에 대한 결과 처리는 이벤트 콜백 함수를 이용합니다.
- loadstart : 요청에 대한 응답을 받기 시작한 경우 발생하는 이벤트
- progress : 요청에 대한 응답을 받는 도중 주기적으로 발생하는 이벤트 예) progress bar
-  load : 요청에 대한 응답이 성공적으로 완료한 경우 발생하는 이벤트
- abort : 요청이 취소된 경우 발생하는 이벤트 실패됐을 때, 무언가를 실행해라
- error : 요청에 에러가 발생한 경우 발생하는 이벤트
- loadend : 성공, 실패와 관련없이 요청이 완료된 경우 발생하는 이벤트
- readystatechange : readystate 값이 변경될 때 발생하는 이벤트

# XMLHttpRequest

- send() 함수에 의한 요청 결과가 전송이 완료된 후 발생하는 load 이벤트를 등록하여 콜백함수에서 XMLHttpRequest의 responseText 프로퍼티로 서버의 데이터를 획득한 코드입니다.
- XMLHttpRequest에서 제공하는 프로퍼티는 아래의 것들이 있습니다.
- status : HTTP Response 상태 코드, 200, 404 등
- readyState : HTTP 요청에 대한 상태
- statusText : HTTP Response의 상태 표시 문자열, OK 등
- responseText : HTTP Response 결과 문자열
- responseType : HTTP Response 결과 타입

(단순 문자열 vs JSON 데이터)  
백엔드 개발자랑 소통하면 알 수 있으나,  
소통하기 어려운 상황에 이 프로퍼티로  
동적으로 알아낼 수 있다.

```
xhr.onload = function(){
  if(xhr.status === 200){
    let data = JSON.parse(xhr.responseText);
    resultNode.innerHTML = data.result;
  }
};
```

[서버에서 넘겨 주는 데이터 형식]

1. 단순 문자열
2. JSON (가장 빈번)
3. XML

JavaScript Object Notation

자바스크립트 객체

JSON.stringify(obj)

JSON 문자열

JSON.parse(JSON문자열)

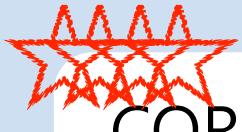
Object literal은 JS만의 독특한 오브젝트 선언법.  
'IT 업계 : data 여러개를 효과적으로 묶어서 전달할 순 없을까?'  
'JS에서 쓰는 Object 표현식과 비슷하게 쓰자!!!'

그러나 JSON은 문자열임.

# XMLHttpRequest

- readyState 프로퍼티의 값은 XMLHttpRequest 객체의 다양한 작업 상태를 표현하며 예를 들어 서버 요청이 된 것인지? 서버에서 응답을 받은 것인지 등 다양한 상태를 파악할 때 이용됩니다.
  - readyState의 값은 정수 값이며 각각의 의미는 다음과 같습니다.
- 0 : UNSET – 아직 open() 으로 초기화 되기 전 상태
  - 1: OPENED – open() 함수에 의해 초기화 되고 아직 서버 요청이 발생하지 않은 상태
  - 2: HEADER\_RECEIVED : 서버로부터 헤더값이 전달된 상태
  - 3: LOADING : 서버 데이터가 전송되고 있는 상태
  - 4: DONE : 모든 작업이 완료된 상태

이걸 확인하면서 프로그램을 짤 수도 있다.



Ajax 통신할 때 에러 상황이 발생할 수 있기 때문에 이 에러가 뭔지는 반드시 알고 있어야 한다.

- CORS 는 Cross Origin Response Sharing 의 약어이며 Ajax 로 요청한 서버의 Origin 이 Ajax 를 요청하는 HTML 의 Origin 과 다른 경우 서버의 응답을 받을 수 없게 됩니다.

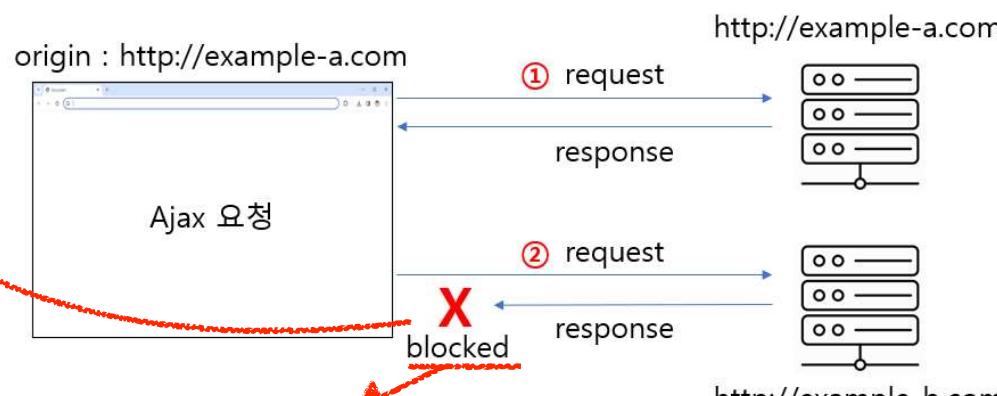
그럼 우리 사이트에서 모든 정보를  
가지고 있는 게 아닌데,  
다른 사이트의 데이터는 요청 못하나?

No. 백엔드에서 응답 할 때  
결과 헤더에 '크로스 오리진 허용'  
을 해주면 프론트에서  
그 데이터를 받을 수 있다.

오리진      **오리진이 다른 경우를 '크로스 오리진'이라고 한다.**

https://developer.mozilla.org:80/ko/docs/Web/API/Storage

프로토콜      도메인      포트      경로



✖ Access to XMLHttpRequest at '<http://localhost:3000/sum/>' from origin '<http://127.0.0.1:5500>' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

Ajax 통신만 그렇다.  
링크로 다른 오리진에 접속하는 건  
에러가 안 뜬다.

왜? Ajax는 개발자 자바스크립트 코드가  
들어가는 것이기 때문에, 특정 서버나  
유저에게 안 좋은 상황이 발생한다.

# CORS

---

- 다른 오리진의 서버와 Ajax 통신이 정상적으로 되려면 Ajax에 의해 요청을 받은 서버 애플리케이션에서 응답 헤더에 설정을 해주어야 합니다.
- 응답 헤더에 Access-Control-Allow-Origin 설정을 통해 오리진이 다르더라도 브라우저에서 blocked 시키지 말고 정상적으로 처리하게 할 수 있습니다.

CORS 문제를 해결하기 위한 서버측 코드

```
1 response.setHeader('Access-Control-Allow-Origin', '*');
2 response.setHeader("Access-Control-Allow-Headers", "X-Requested-
3 With");
```

# CORS

---

- 응답 헤더에 Access-Control-Allow-Origin 을 “ \* ” 로 설정하게 되면 모든 오리진에 의한 요청을 허락하겠다는 의미
  - 특정 오리진의 요청만 허락하고 싶다면 “ \* ” 대신 허락하고자 하는 오리진의 URL 을 써주면 됩니다.
- 
- Access-Control-Allow-Headers 를 이용해 헤더 정보에 어떤 값이 있는 경우에만 허락하겠다고 명시할 수 있으며 “X-Requested-With” 라고 설정하면 Ajax 요청에 한해서 허락하겠다는 의미가 됩니다.
  - Access-Control-Allow-Methods 설정을 통해 특정 HTTP Method 에 한해서만 허락하겠다고 설정할 수도 있습니다.

# Axios

Ajax 코드를 좀 더 편하게 이용할 수 있는 라이브러리들. 최근에는 Axios가 1황

## 과거의 맹주

- jquery ajax, fetch, superagent, Axios 등 다양한 Ajax 라이브러리가 나와 있습니다.

- 브라우저에서는 XMLHttpRequest 이용

내부적으로는 이렇게 동작

- Node.js에서는 http 요청 이용

우리가 굳이 Promise로 묶어서  
할 필요가 없다.  
우리는 바로 .then을 걸어서  
결과값을 받으면 된다.

- Promise API 지원

★ 요청 및 응답 Interceptor 지원

- 요청 및 응답 데이터 자동 변환

이게 필요하다면 Axios가 정말 유용할 수 있다.

실전에서 상당히 유용하다.

서버에서 넘어온 데이터가 json 문자열이어도 자동으로 js object로 전환해준다.

axios CDN 설정

```
1 <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

axios를 쓰려면 CDN을 사용해야 한다.

# Axios

## axios 를 이용한 서버 요청

- axios() 함수를 호출하면서 매개변수로 서버 요청을 위한 정보를 설정해 주어야 합니다.

```
axios({  
  method:'get',  
  url: `http://localhost:3000/sum/${num}`  
})  
.then(response => {  
  printResult(response.data.result)  
})
```

매번 이렇게 하는게 귀찮다?  
아래페이지에 계속…

# Axios

## axios 를 이용한 서버 요청

- axios() 함수의 매개변수에 설정 정보를 config 설정이라고 하며 여러가지 정보를 설정할 수 있습니다.
- config 정보 중 필수는 요청 서버의 URL이며 생략하면 기본으로 GET 방식을 이용하게 됩니다.
- 편리성을 위해서 각각의 HTTP Method 방식으로 요청하는 아래의 함수도 제공합니다.

### 〈 http request method의 header 정보 〉

- axios.request(config)
- axios.get(url[, config])
- axios.delete(url[, config])
- axios.head(url[, config])
- axios.options(url[, config])
- axios.post(url[, data[, config]])
- axios.put(url[, data[, config]])
- axios.patch(url[, data[, config]])

get(default) – 서버야 데이터 줘 (read)

post – 데이터 저장해줘 (create)

put – 데이터 수정해줘 (update)

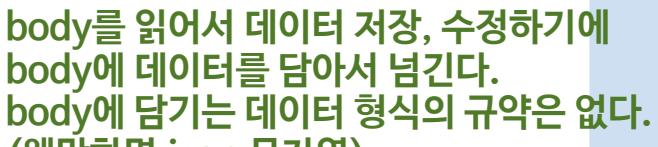
delete – 데이터 삭제해줘 (delete)

...

(patch, head, option)

body를 거의 안 읽어. 하지만  
데이터를 넘기겠다면, header 정보로  
넘기는 방식

- url path에 추가해서 넘기기
- url search 문자열에 추가해서 넘기기



body를 읽어서 데이터 저장, 수정하기에  
body에 데이터를 담아서 넘긴다.  
body에 담기는 데이터 형식의 규약은 없다.  
(웬만하면 json 문자열)

# Axios

## axios 를 이용한 서버 요청

```
axios.get(`http://localhost:3000/sum/${num}`)
  .then(response => {
    printResult(response.data.result)
  })
```

- axios.post(), axios.put(), axios.patch() 함수를 이용한다면 URL 정보 이외에 서버에 바디 스트림으로 전송하는 데이터를 명시해야 하는데 두번째 매개변수로 전송하는 데이터를 지정해 주어야 합니다.

```
axios.post(`http://localhost:3000/post_test`,
{
  name: '홍길동',
  age : 20
})
```

넘기는 데이터

# Axios

## config 설정

- 공통적인 config 가 반복적으로 설정 된다면 한번만 설정하고 재사용 되게 할 수 있습니다.
- 공통으로 들어갈 config 를 설정하는 방법은 axios 객체의 defaults 에 설정하는 방법이 있고, custom axios 객체를 만들면서 공통 정보를 설정하는 방법이 있습니다.

1

- defaults 에 설정하는 방법
- config 설정을 axios.defaults 에 추가하게 되면 이후 axios 로 서버 요청을 할 때 공통으로 적용되게 됩니다.

이 두가지가 서버 요청  
할 때마다 공통적으로  
계속 들어가야 한다면,  
defaults에 설정해두자.

```
axios.defaults.baseURL = "http://localhost:3000/"  
axios.defaults.timeout = 2000
```

```
axios.post(`post_test`,  
{  
    name: '홍길동',  
    age : 20  
})
```

무작정 기다릴 순 없다.  
서버 요청 제한 시간을 건다.

# Axios

기본적으로는 라이브러리에서 제공되는 axios 객체를 사용했는데,  
내가 직접 새로운 커스텀 객체를 만들어서 사용할 수도 있다.  
개발자가 원하는 만큼 만들 수 있다.

## config 설정

- 공통 config 설정을 custom axios 객체를 만들면서 설정할 수도 있습니다.
- custom axios 는 create() 함수로 만들어 집니다.

```
const myAxios = axios.create({  
    baseURL : "http://localhost:3000/",  
    timeout: 2000  
})  
  
myAxios.post(`post_test`,  
{  
    name: '홍길동',  
    age : 20  
})
```



# Axios

## config 설정

- config에 설정될 수 있는 몇몇 정보에 대해 살펴보겠습니다.
- url : 서버 요청 URL
- method : HTTP Request Method
- baseURL : url이 http 혹은 https로 시작하지 않았을 때 url 앞에 들어갈 공통 URL
- data : POST, PUT, PATCH 요청 시 서버에 전송될 바디 스트림 데이터 (get 방식은 안 된다)
- timeout : 요청 타임아웃 시간
- params : 요청 URL 파라미터

url에 뒤에 붙여서 전달하는 데이터

url은 필수 정보이기에, 모든 방식에 적용 가능

하지만 header 정보에 data를 넣어 전송하는 건  
한계가 있다.

- (1) 데이터 사이즈가 크다면 불가능
- (2) 보안을 지켜야 하는 데이터를 url에 넣으면  
누구나 볼 수 있기에 위험함.

```
axios({
  method: 'get',
  url: '^http://localhost:3000/config_test^',
  params: {
    data1: "hello",
    data2: 10
  }
})
```

# Axios

## config 설정

- transformRequest      요청을 할 때, 이 함수를 거쳐서 요청된다. 여러개 쓰면 순서대로 적용.
- transformResponse     응답을 받을 때, 이 함수를 거쳐서 응답을 받는다.
- transformRequest 와 transformResponse 를 등록하여 데이터를 추가하거나 제거하는 등의 작업을 할 수 있다
- 배열에 요청과 응답에 실행되어야 하는 함수를 여러 개 등록할 수 있습니다.
- 여러 함수가 등록되었다면 등록한 순서대로 실행되게 됩니다.

```
transformRequest:[
  function (data, headers) {
    headers["Content-Type"] = "application/json"
    let newData = {...data, key: 1}
    return JSON.stringify(newData);
  }
],
transformResponse:[
  function(data){
    const jsonData = JSON.parse(data);
    let newData = {...jsonData, index: 1}
    return newData
  }
]
```

# Axios

---

## config 설정

- transformRequest 에 등록된 함수를 보면 매개변수에 data 와 header 가 전달되는데 data 는 서버에 전송되는 바디 스트림 데이터입니다.
- 즉 POST, PUT 등 바디 스트림을 이용하는 요청의 데이터를 조작하는데 사용됩니다.
- 또한 header 정보가 함수에 전달됨으로 header 를 추가하는 등의 작업을 할 수 있습니다.
- transformResponse 에 등록된 함수의 매개변수는 서버에서 전송되는 데이터입니다.



# 감사합니다

단단히 마음먹고 떠난 사람은  
산꼭대기에 도착할 수 있다.  
산은 올라가는 사람에게만 정복된다.



윌리엄 셰익스피어

William Shakespeare