자바스크립트 AJAX 요청 방식

정통적으로 XMLHttpRequest() 객체를 생성하여 요청하는 방법이 있지만 문법이 난해하고 가독성도 좋지 않다. 따라서 이번시간에는 자바스크립트 AJAX 통신의 최신 기술인 `fetch()` 메서드 사용법에 대해 알아보는 시간을 가져볼 예정이다.

XML Http Request 방식

xmlhttprequest 객체를 이용한 정통적인 초창기 비동기 서버 요청 방식이다. 성능에는 문제는 없지만 코드가 복잡하고 가독성이 좋지 않다는 단점이 있었다.

JAVASCRIPT

```
var httpRequest = new XMLHttpRequest();
httpRequest.onreadystatechange = function () {
   if (httpRequest.readyState == XMLHttpRequest.DONE && httpRequest.status == 200) {
        document.getElementById("text").innerHTML = httpRequest.responseText;
   }
}
httpRequest.open("GET", "ajax_intro_data.txt", true);
httpRequest.send();
```

Fetch API 방식

이벤트 기반인 XMLHttpRequest과는 달리 fetch API는 Promise 기반으로 구성되어 있어 **비동기 처리 프로그래밍 방식**에 잘 맞는 형태이다. 그래서 then이나 catch 와 같은 체이닝으로 작성할 수 있다는 장점이 있다. 또한 fetch API는 **JS 기본 기능** 이기 때문에, JQuery와 같이 CDN 과 같은 다른 작업을 하지 않아도 바로 사용할 수 있다.

JAVASCRIPT

```
fetch('ajax_intro_data.txt')
    .then( response => response.text() )
    .then( text => { document.getElementById("#t").innerHTML = text; } )
/* 'fetch('서버주소')' 는 웹 브라우저에게 '이 서버주소로 요청해줘' 라는 의미이고,
뒤에 .then이 붙으면 '요청 끝나고나서 이 할일을 해줘!' 라는 것이다. */
```

fetch 문법 사용법

JAVASCRIPT

```
fetch("https://jsonplaceholder.typicode.com/posts", option)
   .then(res => res.text())
   .then(text => console.log(text));
```

fetch 에는 기본적으로 첫 번째 인자에 요청할 url이 들어간다.

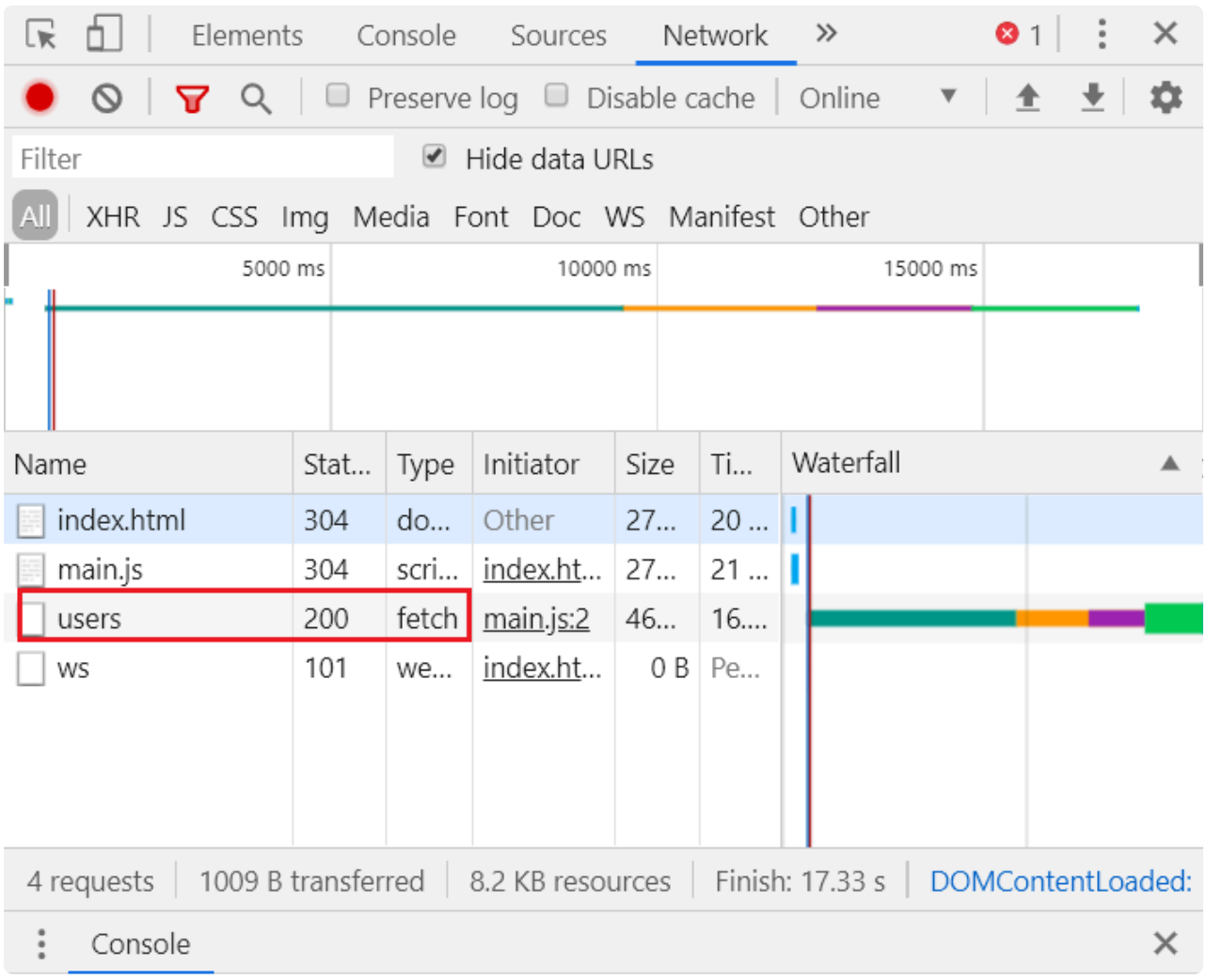
기본적으로 http 메소드 중 GET으로 동작한다.

fetch를 통해 ajax를 호출 시 해당 주소에 요청을 보낸 다음, 응답 객체(Promise object Response)를 받는다.

그러면 첫 번째 then에서 그 응답을 받게되고, res.text() 메서드로 파싱한 text값을 리턴한다.

그러면 그 다음 then에서 리턴받은 text 값을 받고, 원하는 처리를 할 수 있게 된다.

개발자 도구의 네트워크 탭에 가보면 fetch를 통해 얻어온 데이터를 볼수 있다.



fetch의 response 속성

fetch를 통해 요청을 하고 서버로부터 값을 응답 받으면 .then을 통해 함수의 인자에 담기게 되는데 이 값은 **Response객체로서 여러가지 정보**를 담고 있다.
필요한 변수값이나 메서드를 뽑아내서 값을 얻으면 된다.

```
fetch.html:17
Response {type: "basic", url: "http://localhost/ajax/111.txt", redirected: false, status: 200, ok: true, …}
  body: (...)
  bodyUsed: true
  headers: Headers {}
  ok: true
  redirected: false
  status: 200
  statusText: "OK"
  type: "basic"
  url: "http://localhost/ajax/111.txt"
  [[Prototype]]: Response
```

- response.status - HTTP 상태 코드(예: 200)
- response.ok - HTTP 상태 코드가 200과 299 사이일 경우 true
- response.body — 내용
- response.text() - 응답을 읽고 텍스트를 반환한다,
- response.json() - 응답을 JSON 형태로 파싱한다,
- response.formData() - 응답을 FormData 객체 형태로 반환한다.
- response.blob() - 응답을 Blob(타입이 있는 바이너리 데이터) 형태로 반환한다.
- response.arrayBuffer() - 응답을 ArrayBuffer(바이너리 데이터를 로우 레벨 형식으로 표현한 것) 형태로 반환한다.

> Tip
>
> ⚠ 주의
> 응답 자료 형태 반환 메서드는 한번만 사용 할 수 있다.
> 만일 response.text()를 사용해 응답을 얻었다면 본문의 콘텐츠는 모두 처리 된 상태이기 때문에 뒤에 또 response.json() 써줘도 동작하지 않게 된다.

Fetch - CRUD 요청하기

http 요청에는 get, post 뿐만 아니라 put, delete 같은 여러가지가 있다.
fetch() 메소드로 어떻게 http 요청이랑 조합해서 사용하는지 그리고 그 문법을 알아보는 시간을 가져보자.

HTTP Method 종류 (CRUD)

METHOD	역할
POST	POST를 통해 해당 URI를 요청하면 리소스를 생성
GET	GET를 통해 해당 리소스를 조회합니다. 리소스를 조회하고 해당 도큐먼트에 대한 자세한 정보를 가져
PUT	PUT를 통해 해당 리소스를 수정
DELETE	DELETE를 통해 리소스를 삭제

fetch - GET Method

GET : 존재하는 자원을 요청

→ 단순히 원격 API에 있는 데이터를 가져올 때 쓰임

→ fetch함수는 **디폴트로 GET 방식으로 작동**하고, 옵션 인자가 필요 없음.

→ 응답(response) 객체는 json() 메서드를 제공하고, 이 메서드를 호출하면 응답(response) 객체로부터 JSON 형태의 데이터를 자바스크립트 객체로 변환하여 얻을 수 있음.

JAVASCRIPT

```
fetch("https://jsonplaceholder.typicode.com/posts/1") // posts의 id 1인 엘리먼트를 가져옴
  .then((response) => response.json())
  .then((data) => console.log(data))
```

JAVASCRIPT

```
{
  "userId": 1,
  "id": 1,
  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
  "body": "quia et suscipit↵suscipit recusandae consequuntur …strum rerum est autem sunt rem eveniet architecto"
}
```

fetch - POST Method

POST : 새로운 자원 생성 요청

→ 폼 등을 사용해서 데이터를 만들어 낼 때, 보내는 데이터의 양이 많거나, 비밀번호 등 개인정보를 보낼 때 POST 메서드 사용

→ 새로운 포스트 생성 위해서는 **method 옵션을 POST로 지정**해주고, **headers 옵션으로 JSON 포맷 사용한다고 알려줘야 함. body 옵션에는 요청 데이터를 JSON 포맷**으로 넣어줌.

- method - HTTP 메서드
- headers - 요청 헤드가 담긴 객체(제약 사항이 있음)
- body - 보내려는 데이터(요청 본문)로 string이나 FormData, BufferSource, Blob, UrlSearchParams 객체 형태

JAVASCRIPT

```
fetch("https://jsonplaceholder.typicode.com/posts", {
  method: "POST", // POST
  headers: { // 헤더 조작
    "Content-Type": "application/json",
  },
  body: JSON.stringify({ // 자바스크립트 객체를 json화 한다.
    title: "Test",
    body: "I am testing!",
    userId: 1,
  }),
})
  .then((response) => response.json())
  .then((data) => console.log(data))
```

fetch - PUT Method (전체수정)

PUT : 존재하는 자원 변경 요청

→ API에서 관리하는 데이터의 **수정**을 위해 PUT 메서드 사용함.

→ method 옵션만 PUT으로 설정한다는 점 빼놓고는 POST 방식과 비슷

→ **아예 전체를 body의 데이터로 교체해버림.**

JAVASCRIPT

```
fetch("https://jsonplaceholder.typicode.com/posts", {
  method: "PUT",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    title: "Test" // 아예 title 엘리먼트로 전체 데이터를 바꿈. 마치 innerHTML같이.
  }),
})
  .then((response) => response.json())
  .then((data) => console.log(data))
```

fetch - PATCH Method (부분수정)

PATCH : 존재하는 자원 일부 변경 요청

→ body의 데이터와 알맞는 **일부**만을 교체함 .

JAVASCRIPT

```
fetch("https://jsonplaceholder.typicode.com/posts/1", { // posts의 id 1인 엘리먼트를 수정
  method: "PATCH",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    title: "Test" // title만 바꿈. 나머지 요소는 건들지 않음.
  }),
})
  .then((response) => response.json())
  .then((data) => console.log(data))
```

fetch - DELETE Method

DELETE : 존재하는 자원 삭제 요청

→ 보낼 데이터가 없기 때문에 headers, body 옵션이 필요없음

JAVASCRIPT

```
fetch("https://jsonplaceholder.typicode.com/posts/1", {
  method: "DELETE",
})
  .then((response) => response.json())
  .then((data) => console.log(data))
```

Fetch - async / await 문법

fetch의 리턴값 response는 Promise객체 이다. 당연히 await / async 문법으로 보다 가독성을 높이게 코딩 할 수 있다.

JAVASCRIPT

```
fetch("https://jsonplaceholder.typicode.com/posts", option)
.then(res => res.text())
.then(text => console.log(text));
```

JAVASCRIPT

```
(async () => {
  let res = await fetch("https://jsonplaceholder.typicode.com/posts", option);
  let text = await res.text();
  console.log(text);
})() //await은 async함수내에서만 쓸수 있으니, 익명 async 바로 실행함수를 통해 활용합니다.
```

fetch 모듈화 - 사용성 개선하기

fetch() 함수는 사용법이 아주 간단하지만, 계속 사용하다보면 똑같은 코드가 반복된다는 것을 느낄 것이다.

예를 들어, 응답 데이터을 얻기 위해서 response.json()을 매번 호출하거나, 데이터를 보낼 때, HTTP 요청 헤더에 "Content-Type": "application/json"로 일일히 설정해줘야 되거나 써야 될게 많다.

한두번 fetch() 를 사용하는 정도는 문제가 안되지만, 여러번 사용할 일이 있으면 따로 헬퍼 함수를 만들어 모듈화를 해놓는게 나중에 정말 편하게 사용할수 있다.

아래 코드와 같이 POST 요청을 보낼 필요가 있다면 함수로 따로 미리 구성을 짜놓고, 인자로 값들을 전달해 바로바로 값을 응답 받는 식으로 고급 구성을 해놓는다.

JAVASCRIPT

```
async function post(host, path, body, headers = {}) {
  const url = `https://${host}/${path}`;
  const options = {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      ...headers,
    },
    body: JSON.stringify(body),
  };
  const res = await fetch(url, options);
  const data = await res.json();
  if (res.ok) {
    return data;
  } else {
    throw Error(data);
  }
}
post("jsonplaceholder.typicode.com", "posts", {
  title: "Test",
  body: "I am testing!",
  userId: 1,
})
  .then((data) => console.log(data))
  .catch((error) => console.log(error));
```

구형 브라우저에서 fetch 사용하기

fetch는 ES6 최신 자바스크립트 기술이다. 따라서 인터넷 익스플로어 같은 구형 브라우저에는 동작을 안하는데, 이때는 따로 polyfill된 코드를 적용해 같이 사용하면 무리없이 fetch를 사용 할수 있다.

폴리필(polyfill)이라는 단어가 뭔가 생소해서 복잡하고 어려울것 같아보이지만 직접해보면 매우 간단하다.

> **Info**

> 폴리필(polyfill)은 웹 개발에서 기능을 지원하지 않는 웹 브라우저 상의 기능을 구현하는 코드를 뜻한다.

위 링크에서 fetch.js라는 라이브러리를 다운받고, 알맞은 경로로 src를 지정하면 된다. 이렇게 해주면 fetch api를 지원하는 최신브라우저는 그대로 진행하고, 지원하지 않는 구형 브라우저는 fectch.js내에 있는 함수를 통해 ajax가 진행되게 됩니다.

HTML

```
<head>
  <meta charset="utf-8">
  <script src="../fetch.js"></script>
</head>
```