
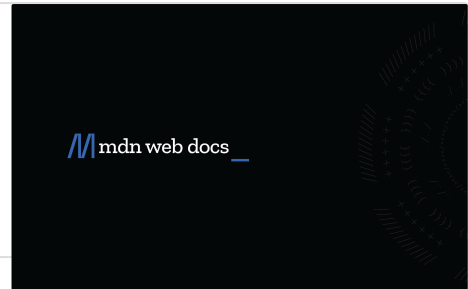


fetch API

Fetch API - Web APIs | MDN

The Fetch API provides an interface for fetching resources (including across the network). It is a more powerful and flexible replacement for XMLHttpRequest.

 https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API



자바스크립트를 사용하면 필요할 때 서버에 네트워크 요청을 보내고 새로운 정보를 받아오는 일을 할 수 있습니다.

네트워크 요청은 다음과 같은 경우에 이뤄집니다.

- 주문 전송
- 사용자 정보 읽기
- 서버에서 최신 변경분 가져오기



AJAX(**A**synchronous **J**avaScript **A**nd **X**ML)는 서버에서 추가 정보를 비동기적으로 가져올 수 있게 해주는 포괄적인 기술을 나타내는 용어입니다.

fetch 사용법

```
fetch(url, options)
  .then((response) => console.log("response:", response))
  .catch((error) => console.log("error:", error));
```

- `fetch()` 함수는 첫번째 인자로 URL, 두번째 인자로 옵션 객체(method나 header등을 지정할 수 있다)를 받고, Promise 타입의 객체를 반환합니다
- 반환된 객체는, API 호출이 성공했을 경우에는 응답(response) 객체를 resolve하고, 실패했을 경우에는 예외(error) 객체를 reject합니다

GET 호출

: 단순히 원격 API에 있는 데이터를 가져올 때 쓰이는 HTTP 통신

- JSON Placeholder라는 인터넷에 공개된 API를 사용해서 예제 코드를 작성
- `fetch()` 함수는 디폴트로 GET 방식으로 작동하고 GET 방식은 요청 전문을 받지 않기 때문에 옵션 인자가 필요가 없습니다.

```
fetch("https://jsonplaceholder.typicode.com/posts/1").then  
((response) =>  
  console.log(response)  
);
```

결과

```
Response {status: 200, ok: true, redirected: false, type:  
"cors", url: "https://jsonplaceholder.typicode.com/posts/  
1", ...}
```

대부분의 REST API들은 JSON 형태의 데이터를 응답하기 때문에, 응답(response) 객체는 `json()` 메서드를 제공합니다

```
fetch("https://jsonplaceholder.typicode.com/posts/1")  
  .then((response) => response.json())  
  .then((data) => console.log(data));
```

결과

응답(response) 객체로 부터 JSON 포맷의 응답 전문을 자바스크립트 객체로 변환하여 얻을 수 있습니다

```
{  
  "userId": 1,  
  "id": 1,
```

```
"title": "sunt aut facere repellat provident occaecati ex  
cepturi optio reprehenderit",  
"body": "quia et suscipit↵suscipit recusandae consequunt  
ur ...strum rerum est autem sunt rem eveniet architecto"  
}
```

단순히 특정 API에 저장된 데이터를 보여주는 웹페이지나 애플리케이션에서는 GET 방식의 HTTP 통신으로 충분합니다

response에는 프라미스를 기반으로 하는 다양한 메서드가 있습니다.

이 메서드들을 사용하면 다양한 형태의 응답 본문을 처리할 수 있습니다.

- **response.text()** – 응답을 읽고 텍스트를 반환합니다,
- **response.json()** – 응답을 JSON 형태로 파싱합니다,
- **response.formData()** – 응답을 FormData 객체 형태로 반환합니다.
- **response.blob()** – 응답을 Blob(타입이 있는 바이너리 데이터 = 이진 데이터) 형태로 반환합니다.
- **response.arrayBuffer()** – 응답을 ArrayBuffer(바이너리 데이터를 로우 레벨 형식으로 표현한 것) 형태로 반환합니다.

이 외에도 response.body가 있는데, ReadableStream 객체인 response.body를 사용하면 응답 본문을 청크 단위로 읽을 수 있습니다.

POST 호출

: 원격 API에서 관리하고 있는 데이터를 생성해야 한다면 요청 전문을 포함할 수 있는 POST 방식의 HTTP 통신이 필요합니다

- **method** 옵션을 **POST** 로 지정해주고, **headers** 옵션을 통해 JSON 포맷을 사용한다고 알려줘야 하며, 요청 전문을 JSON 포맷으로 직렬화하여 가장 중요한 **body** 옵션에 설

정해줍니다.

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

결과

이번에는 응답 객체를 통해 응답 코드가 `201 Created` 인 것을 알 수 있습니다

- 201은 post나 put으로 게시물 작성이나 회원 가입 등의 새로운 데이터를 서버에 생성하는 작업이 성공했을 때 반환 됨

```
Response {type: "cors", url: "https://jsonplaceholder.typicode.com/posts", redirected: false, status: 201, ok: true, ...}
```

응답 객체의 `json()` 메서드를 호출하면 응답 전문을 객체 형태로 얻을 수 있습니다

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

```
    }),  
  })  
  .then((response) => response.json())  
  .then((data) => console.log(data));
```

결과

```
{title: "Test", body: "I am testing!", userId: 1, id: 101}
```

PUT, DELETE 호출

: GET과 POST만큼은 아니지만, 원격 API에서 관리하는 데이터의 수정과 삭제를 해야 할 때 사용합니다.

PUT 방식은 `method` 옵션만 `PUT` 으로 설정한다는 점 빼놓고는 POST 방식과 매우 흡사합니다.

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

결과

```
{title: "Test", body: "I am testing!", userId: 1, id: 1}
```

DELETE 방식에서는 보낼 데이터가 없기 때문에, `headers` 와 `body` 옵션이 필요가 없습니다.

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

결과

```
{}
```

사용성 개선

`fetch()` 함수는 사용법이 아주 간단하지만, 계속 사용하다보면 똑같은 코드가 계속 반복 된다는 것을 알 수 있습니다.

예를 들어, 응답 데이터를 얻기 위해서 `response.json()` 을 매번 호출하거나,

데이터를 보낼 때, HTTP 요청 헤더에 `"Content-Type": "application/json"` 로 설정해주는 부분은 지루하게 느껴질 수 있습니다.

또한 `fetch()` 함수의 Promise 기반의 API가 좀 투박하다고 느끼실 수도 있습니다.

이럴 때는

`fetch()` 함수를 직접 사용하기 보다는, 본인 입맛에 맞게 별도의 함수나 모듈로 빼서 사용하는 것을 추천 합니다.

프로젝트의 상황에 맞게 다음과 같이 `async/await` 키워드를 이용하여 HTTP 방식별로 비동기 함수를 작성하고 모듈화하여 사용 합니다.

```
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));

```