8253jang@daum.net
장희정

학습 목차

1) 동기(Synchronous) vs 비동기(Asynchronous)
2) 자바스크립트에서 비동기 작업이 필요한 이유
3) 비동기 작업과 콜백 함수
4) Promise 객체의 등장
5) async / await 등장
6) Ajax(Asynchronous JavaScript And XML)란?
7) 비동기 통신(Asynchronous Communication)
8) 자바스크립트에서 비동기 통신을 위한 주요 함수/기술
9) JSON
10) GSON Library
11) Ajax TodoList 실습

8253jang@daum.net
장희정

동기(Synchronous) vs 비동기(Asynchronous)

1. 동기(Synchronous)

- **정의**: 작업이 순서대로 하나씩 실행되는 방식.
- 현재 실행 중인 작업이 끝나야만 다음 작업을 실행할 수 있음.
- 코드 실행 흐름이 직선적이라 이해하기 쉽지만, 긴 작업(예: 파일 읽기, 네트워크 요청)이 있으면 프로그램이 멈춘 것처럼 보일 수 있음.

예시 (동기 코드):

```
console.log("1. 첫 번째 작업 시작");

function longTask() {
  // 오래 걸리는 작업 시뮬레이션 (3초)
  const start = Date.now();
  while (Date.now() - start < 3000) {}
  console.log("2. 긴 작업 완료");
}

longTask();
console.log("3. 마지막 작업");
```

실행결과

```
1. 첫 번째 작업 시작
2. 긴 작업 완료
3. 마지막 작업
```

긴 작업이 끝날 때까지 **다음 코드 실행이 막혀 있음(Blocking)**

2. 비동기(Asynchronous)

- **정의**: 작업을 요청해두고, 해당 작업이 끝나길 기다리지 않고 **다음 코드부터 실행하는 방식**.
- 긴 작업은 **백그라운드(Web APIs, Node.js API 등)에서 처리되고**, 완료되면 콜백 큐(Task Queue)를 통해 실행됨.
- **자바스크립트는 싱글 스레드지만**, 이벤트 루프(Event Loop)를 이용해 **비동기 동작을 지원**.

예시 (비동기 코드):

```
console.log("1. 첫 번째 작업 시작");

setTimeout(() => {
  console.log("2. 긴 작업(비동기) 완료");
}, 3000);

console.log("3. 마지막 작업");
```

실행 결과:

```
1. 첫 번째 작업 시작
3. 마지막 작업
2. 긴 작업(비동기) 완료
```

👉setTimeout은 백그라운드에 맡기고, **다음 코드가 바로 실행**됨.

3. 비교 정리

구분	동기(Synchronous)	비동기(Asynchronous)
실행 방식	순차적으로 한 줄씩 실행	작업을 맡기고, 끝나면 나중에 실행
흐름	직선적, 이해하기 쉬움	병렬적으로 동작하는 것처럼 보임
장점	코드가 직관적, 디버깅 쉬움	UI 멈춤 방지, 성능 개선
단점	긴 작업 시 전체가 멈춤	코드 구조 복잡 (콜백, 프로미스 필요)
예시	반복문, 수학 계산	`setTimeout`, `fetch`, 이벤트 핸들러

자바스크립트에서 비동기 작업이 필요한 이유

(1) 싱글 스레드 특성

- JS 메인 스레드는 한 번에 한 작업만 실행한다.
- 만약 ,시간이 오래 걸리는 작업(파일 읽기, 서버 요청, 이미지 처리 등)을 동기적으로 실행하면, 그동안 UI가 멈추고 다른 코드도 실행되지 않음 → 사용자 경험이 나빠짐.

(2) Non-blocking(비차단) 실행 필요

- 비동기는 오래 걸리는 작업을 백그라운드(Web API)에 맡겨두고, 메인 스레드는 다른 일을

계속 할 수 있게 해준다.

- 작업이 끝나면 이벤트 루프(Event Loop)가 결과를 가져와 실행한다.

(3) 실제 웹/서버 환경에서 필수

- 웹 브라우저: 버튼 클릭 이벤트, 서버 API 요청, 이미지 로딩 등은 언제 끝날지 모르므로 비동기로 처리해야 한다.

정리하면,

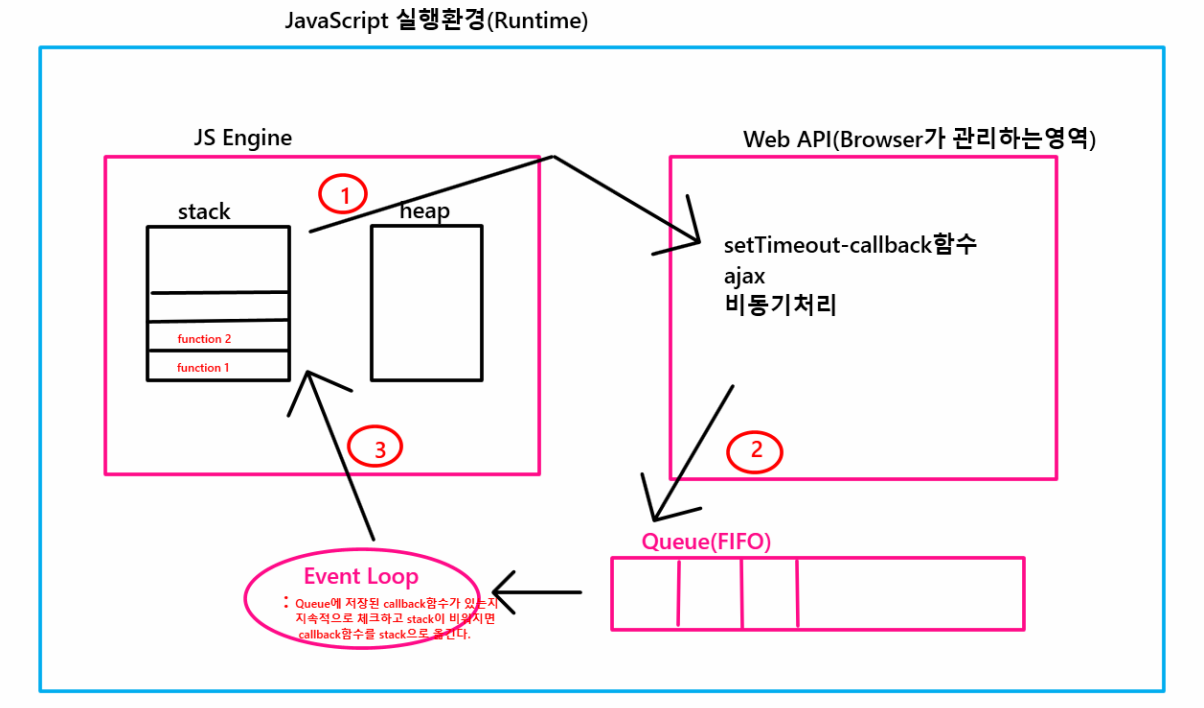
- JS는 싱글 스레드라 오래 걸리는 작업을 동기적으로 실행하면 전체가 멈춤.
- 네트워크 요청, 파일 처리, 타이머, 이벤트 같은 작업은 언제 끝날지 예측 불가 → **비동기로 처리해야 UI/프로그램이 끊기지 않음**.

✅핵심 비유

- **동기**: 음식점에서 한 명이 음식을 다 만들 때까지 줄 서서 기다림.
- **비동기**: 주문만 하고 다른 일 하다가, 음식 나오면 알림 받고 먹음.

그렇다면,

JavaScript가비동기처리를어떻게?



8253jang@daum.net
장희정

비동기 작업과 콜백 함수

💡 왜 콜백이 필요할까?

- 자바스크립트는 싱글 스레드 → 오래 걸리는 작업은 비동기로 처리해야 함.
- **비동기 작업이 끝난 뒤 실행할 코드를 미리 등록**해놓아야 하는데, 이때 사용하는 것이 **콜백 함수(callback function)**.

예시

```
console.log("데이터 요청 시작");

setTimeout(() => {
  console.log("데이터 응답 도착!");
}, 2000);

console.log("다른 작업 실행 중...");
```

설명:

setTimeout 안에 들어간 함수 → **콜백 함수**

비동기 작업이 끝난 시점에 실행되도록 예약

💡콜백 지옥 (Callback Hell)

문제 발생

비동기 작업을 순서대로 실행하려면, 콜백 안에 또 콜백... 을 계속 중첩해야 함.

→ 코드가 들여쓰기 지옥처럼 깊어지고, 가독성/유지보수가 어려워짐.

예시)

```
getUser(1, (user) => {
  console.log("사용자:", user);
  getPosts(user.id, (posts) => {
    console.log("게시물:", posts);
    getComments(posts[0].id, (comments) => {
      console.log("댓글:", comments);
      // 계속 중첩…
    });
  });
});
```

8253jang@daum.net
장희정

문제점:

코드 들여쓰기 너무 깊어짐
에러 처리 복잡 (try/catch 불가, 콜백마다 에러 핸들링)
유지보수 어려움

JavaScript에서 콜백지옥을 벗어나기 위해서 Promise객체 탄생했다.

Promise 객체의 등장

비동기작업을 효율적으로 처리 할수 있도록 도와주는 자바스크립트 내장객체
비동기 작업의 **결과(성공/실패)를 약속(Promise)하는 객체**

상태 3가지:

- **pending**: 대기 중
- **fulfilled**: 성공(resolve)
- **rejected**: 실패(reject)

장점 : 비동기작업 실행, 비동기작업 상태관리, 비동기작업 결과 저장등

MDN
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

w3schools
https://www.w3schools.com/js/js_promise.asp

8253jang@daum.net
장희정

<Promise문법>

Promise Syntax

```
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject();  // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

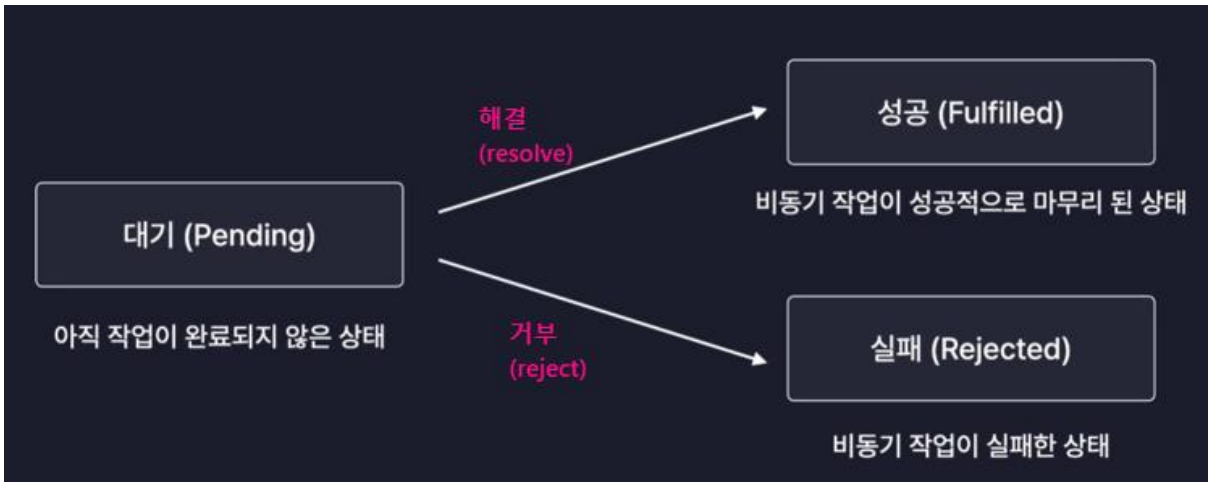
Promise 기본 예시

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("데이터 응답 도착!");  // 성공
    // reject("에러 발생!");       // 실패
  }, 2000);
});

promise
  .then(result => console.log(result)) // 성공 처리
  .catch(error => console.error(error)) // 실패 처리
  .finally(() => console.log("작업 종료"));
```

8253jang@daum.net
장희정

Promise의 3가지 상태



Promise로 콜백 지옥 해결하기

Promise 체이닝

- **콜백 중첩 대신, .then() 체인으로 가독성 있게 작성 가능.**

예시 (콜백 지옥 → Promise 변환)

```
getUser(1)
  .then(user => {
    console.log("사용자:", user);
    return getPosts(user.id);
  })
  .then(posts => {
    console.log("게시물:", posts);
    return getComments(posts[0].id);
  })
  .then(comments => {
    console.log("댓글:", comments);
  })
  .catch(error => {
    console.error("에러:", error);
  });
```

8253jang@daum.net
장희정

장점:

- 들여쓰기 평평해짐 (가독성 ↑)
- 에러는 한 곳에서 처리 가능 (catch)
- 순차 실행이 직관적

then 체이닝 방식의 장점

1. **비동기 작업 흐름 제어 가능**
    - Promise.then()을 통해 비동기 작업(taskA → taskB → taskC)을 순차적으로 처리할 수 있다.
    - 콜백 헬(callback hell)을 어느 정도 개선한 구조
2. **에러 처리 일관성**
    - .catch() 하나로 전체 체인의 오류를 한 번에 처리할 수 있어 try-catch보다 간편한 측면이 있다.
3. **분기 처리 가능**
    - .then() 체이닝 안에서 조건 분기, 중간 로직 삽입 등이 유연하게 가능하다.

then 체이닝 방식의 단점

1. **가독성 저하 (중첩 구조)**
    - 체이닝이 길어질수록 코드의 흐름이 수직적으로 길어져 **가독성이 떨어진다.**
    - 특히 resultA → taskB() → resultB → taskC() ... 이런 구조는 초심자에게 혼란을 준다.
2. **디버깅 불편**
    - 각 .then() 구문에서 에러가 발생했을 때, 어느 부분에서 문제가 생겼는지 **트래킹 하기 어려운 경우**가 많다.
3. **동일한 인자 흐름이 번거로움**
    - 이전 결과(resultA)를 다음 함수에 넘겨주려면 계속 return으로 연결해야 하므로 **로직 흐름이 단절되어 보일 수 있다.**

then() 체이닝 방식은 과거 콜백 지옥을 벗어나기 위해 등장한 Promise의 장점 중 하나이다.
하지만 체인이 길어질수록 코드가 복잡해지고 가독성이 떨어진다.
그래서 이를 **더 간결하고 동기적인 코드처럼** 쓸 수 있도록 도와주는 것이 바로 **async/await 문법**이다.

8253jang@daum.net
장희정

async / await 등장

1. 배경과 등장 이유

- **콜백 함수**: 비동기 제어 가능하지만, 콜백 지옥 문제 발생
- **Promise**: 체이닝(.then())으로 해결했지만, 여전히 코드가 장황하고 동기 코드처럼 직관적이지 않음
- ***그래서,* ES2017(ES8)에서 async/await 문법 등장**
  → 비동기 코드를 동기 코드처럼 읽고 쓸 수 있게 개선

2. async/await의 장점

1. 가독성
   - 중첩된 .then() 체인 없이 동기 코드 같은 흐름으로 작성 가능
2. 에러 처리 간단
   - try...catch로 동기 코드처럼 에러 처리
3. 디버깅 쉬움
   - .then() 체인보다 스택 추적이 직관적
4. Promise 기반이라 호환성 좋음
   - 기존 Promise 함수를 그대로 활용 가능

async& await

async : 어떤 함수를 비동기 함수로 만들어주는 키워드로
함수가 promise를 반환하도록 변환 해주는 키워드

await : async함수 내부에서만 사용이 가능한 키워드
비동기 함수가 다 처리 되기를 기다리는 역할

사용법

(1) async 함수

- 함수 앞에 async를 붙이면, 항상 **Promise를 반환**하는 함수가 됨

```
async function hello() {
  return "Hello";
}

hello().then(msg => console.log(msg)); // Hello
```

(2) await 키워드

- await는 **Promise가 처리될 때까지 기다린 뒤 결과 반환**
- 오직 async 함수 안에서만 사용 가능

```
function delay(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

async function run() {
  console.log("작업 시작");
  await delay(2000); // 2초 기다림
  console.log("2초 후 실행");
}

run();
```

(3) 에러 처리

- .catch() 대신 try...catch 블록 사용

```
function failTask() {
  return new Promise((_, reject) => {
    reject("에러 발생!");
  });
}

async function run() {
  try {
    const result = await failTask();
    console.log(result);
  } catch (error) {
    console.error("잡은 에러:", error);
  }
}

run();
```

(4) 여러 비동기 작업 동시에 처리

- 순차 실행 대신 Promise.all과 함께 사용 가능

```
async function run() {
  const [a, b] = await Promise.all([
    fetch("/data1.json").then(r => r.json()),
    fetch("/data2.json").then(r => r.json())
  ]);
  console.log("data1:", a);
  console.log("data2:", b);
}
```

Promise.all()은 **여러 개의 Promise(비동기 작업)** 를 동시에 실행하고, **모든 Promise가 완료될 때까지 기다린 뒤** 결과를 한꺼번에 반환하는 정적 메서드

async/await 방식의 장점

1. **가독성이 뛰어남 (동기식처럼 보이는 비동기)**
   - await를 사용하면 마치 순차적으로 실행되는 일반 함수처럼 읽히므로 **가독성이 훨씬 좋아진다.**
   - 복잡한 .then() 체이닝보다 **눈에 확 들어오는 코드 흐름**을 제공한다.

```
const resultA = await taskA(3, 4);
const resultB = await taskB(5);
const resultC = await taskC(7);
```

2. **디버깅이 쉬움**
   - 에러가 발생해도 try-catch 블록 안에서 처리할 수 있어 **문제 지점을 정확히 추적**할 수 있다.
   - 디버깅 시 브레이크포인트 걸기도 편하다.
3. **코드 구조 단순화**
   - 중간에 return, .then() 연결 없이도 순차 흐름만으로 동작하므로 **불필요한 구조적 복잡성이 줄어든다.**
   - 비동기 연산 간 의존관계가 명확하게 드러난다.

await는 async 함수 안에서만 사용 가능

- await를 쓰기 위해 반드시 함수 앞에 async를 붙여야 하는 제약이 있다.

구분	콜백	Promise	async/await
코드 스타일	중첩(Callback Hell)	체인 `.then()`	동기 코드처럼 깔끔
에러 처리	콜백 안에서 따로 처리	`.catch()`	`try...catch`
가독성	❌	△	✅
디버깅 편의	❌	△	✅
실무 사용	거의 사용 안 함	자주 사용	현재 표준, 가장 많이 사용

핵심 비유

- **콜백**: "전화 줄 테니 다 끝나면 나 불러"
- **Promise**: "작업이 끝나면 약속대로 알려줄게(.then)"
- **async/await**: "작업 끝날 때까지 잠깐 기다릴게(await), 그리고 나서 이어서 진행하자"

Ajax(Asynchronous JavaScript And XML)란?

Ajax는 특정 프로그래밍 언어나 프레임워크가 아니라, **웹 애플리케이션에서 클라이언트와 서버 간 데이터를 비동기적으로 주고받는 기술적 개념**을 말한다.
Ajax의 핵심은 **웹 페이지 전체를 새로고침하지 않고도 일부 데이터를 갱신할 수 있다**는 점이다. 최근의 웹 애플리케이션은 한 페이지에서 여러 요청을 동시에 처리하고 그 결과를 빠르게 반영해야 하는 경우가 많다. 만약 특정 요청이 완료될 때까지 다른 요청을 진행할 수 없다면, 사용자 입장에서는 로딩이 길어지고 화면 렌더링이 지연되어 불편함을 느끼게 된다.

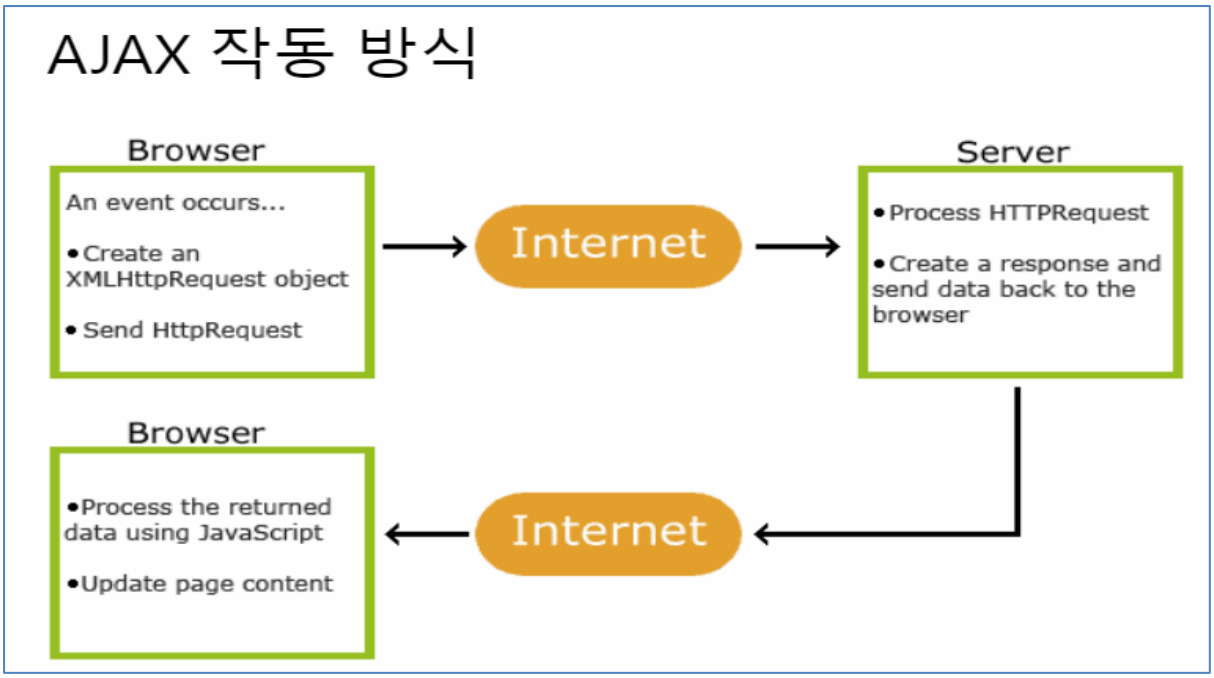
이러한 문제를 해결하기 위해 **Ajax 기술**을 활용하면, 페이지 전체를 새로고침하지 않고도 필요한 부분만 선택적으로 갱신할 수 있다. 이를 통해 웹 페이지는 더 빠르고 효율적으로 동작하며, 사용자는 끊김 없는 자연스러운 경험을 할 수 있다.

브라우저에 내장된 **XMLHttpRequest 객체**(또는 최근에는 fetch API)를 사용하여 서버와 데이터를 주고받으며, 응답 데이터는 **XML뿐만 아니라 JSON, HTML, 텍스트 등 다양한 형식**을 활용할 수 있다. 받은 데이터를 자바스크립트로 가공해 화면 일부에 즉시 반영함으로써, 동적인 웹 애플리케이션을 구현할 수 있다.

8253jang@daum.net
장희정

핵심 정리

- Ajax는 **기술적 개념**이지, 독립된 언어가 아님
- **비동기 통신**을 통해 페이지 전체 새로고침 없이 데이터 일부 갱신 가능
- 주로 **XMLHttpRequest / fetch**를 사용
- 데이터 형식: XML, JSON(가장 많이 사용), 텍스트, HTML 등
- 대표적인 사용 **예:** 실시간 검색, 무한 스크롤, 댓글 등록 등



비동기 통신(Asynchronous Communication)

정의

- **비동기 통신**이란, 클라이언트(브라우저)와 서버 간에 데이터를 주고받을 때 **응답을 기다리며 프로그램 실행을 멈추지 않고**, 그 사이에도 다른 작업을 계속 진행할 수 있는 방식.
- 즉, 요청을 보낸 뒤 서버 응답이 오기 전까지도 웹 페이지는 멈추지 않고 동작하며, 응답이 도착하면 미리 등록된 콜백이나 프로미스를 통해 결과를 처리한다.

장점:

- 페이지 전체 새로고침 없이 일부 데이터만 갱신 가능
- 사용자 경험(UX) 향상 → 끊김 없는 인터랙션 제공

- 서버 자원 및 네트워크 효율 개선

☞ 자바스크립트에서 비동기 통신을 위한 주요 함수/기술

1) XMLHttpRequest (XHR)

- 전통적인 Ajax 구현 방식
- 브라우저 내장 객체로, 서버에 HTTP 요청을 보내고 응답을 받을 수 있음
- 콜백 기반으로 동작하기 때문에 코드가 복잡해지기 쉬움

```
const xhr = new XMLHttpRequest();
xhr.open("GET", "data.json");
xhr.onload = () => {
  if (xhr.status === 200) {
    console.log("응답:", xhr.responseText);
  }
};
xhr.send();
```

2)fetch() API

- ES6(2015) 이후 등장, XMLHttpRequest를 대체하는 표준 API
- Promise 기반 → 가독성이 좋고, async/await와 함께 사용 가능
- JSON 처리도 간단함

```
fetch("data.json")
  .then(response => response.json())
  .then(data => console.log("응답:", data))
  .catch(error => console.error("에러:", error));
```

async/await 버전:

8253jang@daum.net
장희정

```
async function loadData() {
  try {
    const response = await fetch("data.json");
    const data = await response.json();
    console.log("응답:", data);
  } catch (err) {
    console.error("에러:", err);
  }
}
loadData();
```

3) 외부 lib 사용(Promise기반의 Http 비동기 통신라이브러리)

- jQuery lib
- Axios lib

비동기 데이터 요청 방식 비교

구분	AJAX	fetch	async/await	Axios
기본 설명	JavaScript 에서 서버와 통신할 수 있게 해주는 오래된 방식	최신 표준 API 로, Promise 기반 비동기 통신 방식	fetch + Promise 를 더 간결하게 사용하는 async 함수 형태	Promise 기반의 HTTP 요청 라이브러리, 간결하고 직관적
기본 문법	$.ajax({...}) 또는 XMLHttpRequest	fetch(url).then(...).catch(...)	const data = await fetch(url)	axios.get(url).then(...).catch(...)

코드 예시	$.ajax({
	url: 'url',
	method: 'GET',
	success: function(res){
	console.log(res);
	}
	error: function(err) {
	console.error(err);
	}
	});
		fetch('url')
		.then(res => res.json())
		.then(data =>
		console.log(data))
		.catch(err =>
		console.error(err));
			async function getData() {
			try {
			const res = await fetch('url');
			const data = await res.json();
			console.log(data);
			} catch(err) {
			console.error(err);
			}
			}
				axios.get('url')
				.then(res => console.log(res.data))
				.catch(err => console.error(err));

장점	jQuery 사용 시 간편, 오래된 브라우저 호환성 좋음	가독성 향상, 간단한 문법	동기 코드처럼 작성 가능해 가독성 최고	자동 JSON 변환, 응답 간결화, HTTP 메서드 직관적
단점	jQuery 의존성 또는 verbose 한 XML 방식	예외 처리 등 다소 복잡할 수 있음	async, await, try-catch 문법에 익숙해야 함	별도의 라이브러리 설치 필요, 브라우저 환경에서 사용 시 CDN 필요
사용 시기	jQuery 를 이미 쓰고 있다면 적합	현대 브라우저에서 간단한 API 호출	복잡한 비동기 로직을 간결하게 처리할 때	브라우저와 Node.js 모두에서 HTTP 요청을 간편하게 할 때

fetch함수 - 서버와 **HTTP 요청/응답을 주고받기 위한 비동기 함수**

fetch(url, options)

fetch() 함수의 두 번째 인자인 options 객체는 HTTP 요청을 **자세히 설정**할 수 있는 부분으로 이 객체를 통해 **요청 방식(method)**, **헤더(headers)**, **요청 데이터(body)** 등을 지정할 수 있다.

8253jang@daum.net
장희정

특징	설명
✅ Promise 기반	`.then()` 또는 `async/await`로 비동기 처리
✅ 간결한 문법	코드가 짧고 직관적
✅ JSON 처리 편리	응답을 `.json()`으로 쉽게 변환 가능
✅ 기본적으로 GET 방식	`options`를 주면 POST, PUT 등 가능
⚠️ 에러처리는 명시적으로 해야 함	HTTP 상태코드 오류(404, 500)는 `catch`가 잡지 않음 – 수동 체크 필요

옵션	설명	예시
`method`	HTTP 요청 방식	`"GET"`, `"POST"`
`headers`	요청 헤더	`"Content-Type"`
`body`	요청 데이터	`JSON.stringify(data)`
`mode`	CORS 처리	`"cors"`, `"same-origin"`
`credentials`	쿠키 포함 여부	`"include"`
`cache`	캐시 정책	`"no-cache"`
`redirect`	리다이렉트 처리	`"follow"`
`referrer`	참조 헤더 설정	`"no-referrer"`

예시)

```
fetch("https://api.example.com/save", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"  // JSON 형식 명시
  },
  body: JSON.stringify({ name: "Grace", age: 30 })
})
  .then(response => response.json())
  .then(result => console.log("결과:", result))
  .catch(error => console.error("에러:", error));
```

async / await 활용

```
async function loadData() {
  try {
    const response = await fetch("https://api.example.com/data");
    if (!response.ok) throw new Error("에러코드: " + response.status);
    const data = await response.json();
    console.log("데이터:", data);
  } catch (error) {
    console.error("에러 발생:", error);
  }
}
```

JSON이란?

JSON (JavaScript Object Notation)은**데이터를 저장하고 전송할 때 사용하는 경량의 데이터 형식으로 텍스트 기반**이기 때문에 사람이 읽기 쉽고, 기계도 분석하기 쉽다.
웹 클라이언트(브라우저)와 서버 간의 데이터 교환에 자주 사용된다.

8253jang@daum.net
장희정

특징	설명
구조화된 데이터 표현	`{}`(객체), `[]`(배열)을 사용하여 데이터를 구조화
키-값 쌍	`"key": "value"` 형태로 데이터를 저장
가볍고 빠름	XML보다 간단하고 가벼워서 속도가 빠름
범용성	JavaScript뿐만 아니라 Python, Java, PHP 등에서도 쉽게 파싱 가능

<JSON예시>

```
{
  "name": "Grace",
  "age": 30,
  "isInstructor": true
}
```

```
{
  "students": [
    {
      "name": "Minjun",
      "age": 22
    },
    {
      "name": "Jiyoon",
      "age": 24
    }
  ]
}
```

JSON 사용하는 예 (JavaScript)

```
const jsonStr = '{"name": "Grace", "age": 30}';

// JSON 문자열 → 자바스크립트 객체로 변환
const obj = JSON.parse(jsonStr);
console.log(obj.name); // Grace

// 자바스크립트 객체 → JSON 문자열로 변환
const newJsonStr = JSON.stringify(obj);
console.log(newJsonStr); // {"name":"Grace","age":30}
```

JSON.stringify() / JSON.parse()

stringify()	자바스크립트의 값을 JSON 문자열로 변환한다.
	var obj = { name: "John", age: 30, city: "New York" };

8253jang@daum.net
장희정

	var myJSON = JSON.stringify(obj); myJSON// "{"name":"John","age":30,"city":"New York"}"

parse()	JSON 문자열을 자바스크립트 객체로 변환하여 반환한다..
	var txt = '{"name":"John", "age":30, "city":"NewYork"}'; var obj = JSON.parse(txt); obj // {name: "John", age: 30, city: "New York"}

예)ex07-fetch.html

```
<script>
  //1.
// let response =  fetch("https://jsonplaceholder.typicode.com/users"); //Promise반환
//console,console.log((response));



//2.
/* let response = fetch("https://jsonplaceholder.typicode.com/users")
                  .then((res)=>{
                    console.log(res);
                    const data = res.json();
                    console.log("data = " , data);

                  }) //Response객체 반환
                  .catch((err)=>{console.log(err)});*/

  //console.log(response); //fetch가 비동기함수이므로 Promise를 먼저 출력하고 위 console 출력


//3.
/*const getData = async()=>{
  let response = await fetch("https://jsonplaceholder.typicode.com/users")
  console.log(response);
  let data = await response.json();
  console.log(data);
  data.forEach(element => {
      console.log(element.id)
  });
}
getData();*/

//4.예외처리
const getData = async()=>{
  try{
    let response = await fetch("https://jsonplaceholder.typicode.com/users")
    let data = await response.json();
    console.log(data);
  }catch(err){
    console.log(err)
  }
}
getData();
</script>
```

8253jang@daum.net
장희정

fetch()로 비동기 통신할 때 body에 넣을 수 있는 데이터 타입

1. URLSearchParams

정의

- URLSearchParams는 URL의 쿼리 문자열을 다루기 위한 내장 객체
- 키-값 쌍(key=value) 형식으로 데이터를 관리하며, application/x-www-form-urlencoded 방식으로 직렬화

왜 사용하는가?

- 서버가 application/x-www-form-urlencoded 형식 데이터를 요구할 때
- 예전 HTML <form method="post">의 기본 인코딩과 동일한 방식
- 간단한 텍스트 기반 데이터 전송에 적합

예시

```
const params = new URLSearchParams();
params.append("username", "kim");
params.append("age", 30);

fetch("/login", {
  method: "POST",
  headers: {
    "Content-Type": "application/x-www-form-urlencoded"
  },
  body: params
});
```

전송되는 데이터:

```
username=kim&age=30
```

2. FormData

8253jang@daum.net
장희정

정의

- FormData는 HTML <form> 데이터를 쉽게 다루고 전송할 수 있는 객체이다.
- 파일 업로드 같은 이진 데이터(binary data)도 함께 전송 가능.
- 자동으로 multipart/form-data 형식으로 인코딩된다.

왜 사용하는가?

- 이미지, 동영상, 파일 등 대용량/이진 데이터 업로드에 필요
- input[type=file] 값을 포함한 폼 전체를 직렬화해 보내고 싶을 때
- 서버가 multipart/form-data 요청을 처리할 수 있을 때

예시

```
const formData = new FormData();
formData.append("username", "kim");
formData.append("profileImg", fileInput.files[0]); // 파일 업로드

fetch("/upload", {
  method: "POST",
  body: formData // Content-Type 자동 지정됨
});
```

전송되는 데이터 (multipart 형식):

```
------WebKitFormBoundary...
Content-Disposition: form-data; name="username"

kim
------WebKitFormBoundary...
Content-Disposition: form-data; name="profileImg"; filename="me.png"
Content-Type: image/png

(바이너리 데이터)
------WebKitFormBoundary...
```

3. JSON 전송

특징

- 데이터 형식: application/json
- 객체를 문자열(JSON)로 변환해서 전송
- SPA, REST API에서 가장 일반적
- 서버에서 파싱: request.getReader() → JSON 파싱 (Java: Gson, Jackson 등)

예시

```
fetch("/api/user", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"   // 반드시 지정
  },
  body: JSON.stringify({
    username: "kim",
    age: 30
  })
});
```

전송 데이터:

```
{"username":"kim","age":30}
```

비교 정리

구분	JSON	URLSearchParams	FormData
Content-Type	`application/json`	`application/x-www-form-urlencoded`	`multipart/form-data` (자동)
데이터 형식	JSON 문자열	`key=value`	키-값 + 파일(이진)
사용 용도	REST API, SPA, 대부분의 현대 API	전통적인 form 제출, 로그인, 검색	파일 업로드, 이미지/동영상 전송
서버 처리	JSON 파서 필요 (`Gson`, `Jackson`)	`request.getParameter()`	`request.getPart()`, Multipart 처리
장점	구조적 데이터, 계층 표현 가능	간단하고 호환성 높음	파일 업로드 가능, 유연
단점	서버 JSON 파서 필요	복잡한 데이터 표현 어려움	텍스트 전송만이라면 오버헤드 큼

핵심 요약

- JSON → REST API 기본, 복잡한 데이터 구조 전송에 최적
- URLSearchParams → 텍스트 기반 간단 요청 (로그인, 검색)
- FormData → 파일 업로드나 폼 전체 전송 시 사용

8253jang@daum.net
장희정

Gson이란?

Gson(Google + JSON)은 Google이 만든 **자바용 JSON 직렬화/역직렬화 라이브러리이다**.

- **직렬화(Serialize)**: Java 객체 → JSON 문자열
- **역직렬화(Deserialize)**: JSON 문자열 → Java 객체

언제 필요한가?

- Servlet/Tomcat에서 응답 바디를 JSON으로 내려줄 때
- 프런트에서 보낸 JSON 요청 바디를 Java 객체로 파싱할 때
- Ajax/Fetch API와 통신 시, DTO ↔ JSON 변환이 필요할 때
- 설정이 단순하고, Lombok/레코드/POJO 기반으로 빠르게 API를 만들 때

예시) 자바 객체 → JSON 변환 (직렬화)

```
import com.google.gson.Gson;

class User {
    String name;
    int age;
}

public class Main {
    public static void main(String[] args) {
        User user = new User();
        user.name = "Grace";
        user.age = 25;

        Gson gson = new Gson();
        String json = gson.toJson(user);
        System.out.println(json); // {"name":"Grace","age":25}
    }
}
```

예시) JSON → 자바 객체 변환 (역직렬화)

```
String json = "{\"name\":\"Grace\",\"age\":25}";
User user = new Gson().fromJson(json, User.class);
System.out.println(user.name); // Grace
```

8253jang@daum.net
장희정

실습(Front + Back연결)

ex01_ajaxTest.html

```html
<h1> 비동기통신 : javaScript가 제공하는 fetch함수 알아보기 </h1>
<form id="f" method="post">
    이름 : <input type="text" id="name" name="name"><br>
    아이디 : <input type="text" id="id" name="id"><br>

  <button id="btn" type="button">클릭</button>
</form>

<hr>
<div id="result"></div>
</body>
```

```html
<script type="text/javascript">
   document.getElementById("btn").addEventListener("click", async function () {
     const form = document.getElementById("f");
     const formData = new FormData(form);
     //const body = new URLSearchParams(formData);

    try {
        const response = await fetch("ajaxCheck", {
            method: "post",
            body : new URLSearchParams(formData)
        });

        if (!response.ok) {
            throw new Error("서버 응답 오류: " + response.status);
        }

        const resultText = await response.text();
        document.getElementById("result").innerHTML = resultText;

        } catch (error) {
            console.error("Fetch 요청 실패:", error);
        }
});

</script>
```

AjaxTestServlet.java

```java
@WebServlet(urlPatterns = "/ajaxCheck" , loadOnStartup = 1)
public class AjaxTestServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        //전송된 데이터 받기
        String name = req.getParameter("name");
        String id = req.getParameter("id");

        System.out.println("name = " + name);
        System.out.println("id = " + id);

        //서비스 -> dao -> 결과를 받아서
        //브라우져로 응답
        PrintWriter out = resp.getWriter();
        out.println(name+"님 반가워요~~~, Nice meet you");

    }
}
```

8253jang@daum.net
장희정

Ex03_ajax-gson-test.html

```html
<body>
<h3> Ajax Gson Test</h3>
<input type="button" value="text결과" id="ajaxBtn">
<input type="button" value="json결과" id="jsonBtn">

<input type="button" value="DTO결과" id="dtoBtn">
<input type="button" value="List결과" id="listBtn">
<input type="button" value="Map결과" id="mapBtn">

<input type="button" value="textJson결과" id="textJsonBtn">

<hr>
<div id="display"></div>
```

1) text결과

```javascript
const baseURL = `${pageContext.request.contextPath}`; // 서버에서 EL 처리됨

document.getElementById("ajaxBtn").addEventListener("click", async function () {
    try {
        const response = await fetch(`${"${baseURL}"}/ajax`, {
            method: "POST",
            body:new URLSearchParams({//폼데이터를 서버에 전송할때 URLSearchParams는 key=todoList&methodName=selectAll 형태로 만들어줌(서버가 전통적인 방식으로 파라미터 받을 때 사용)
                key: "text",
            })
        });

        if (!response.ok) throw new Error("POST 요청 실패");

        const result = await response.text();
        console.log(result);
    } catch (error) {
        console.log("에러:", error);
    }
});
```

```java
    /**
     * text 결과
     * */
    public void text(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //service---

        //응답
        PrintWriter out = response.getWriter();
        out.println("Welcome to ~");

    }
```

2)Json 결과

```
// JSON 배열 (예: 메뉴 목록)
document.getElementById("jsonBtn").addEventListener("click", async function () {
    try {
        const response = await fetch(`${"${baseURL}"}/ajax?key=json`);
        if (!response.ok) throw new Error("JSON 요청 실패");

        const result = await response.json();
        console.log(result);

        let str = "";
        result.forEach((menu, index) => {
            str += `<input type='checkbox' value='${"${index}"}'>${"${menu}"}`;
        });

        document.getElementById("display").innerHTML = str;
    } catch (error) {
        console.log("에러:", error);
    }
});
```

```
    /**
     * json결과
     * */
    public void json(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //service -> dao
        List<String> menus = Arrays.asList("짜짱면","짬뽕","짬짜면","탕수육","우동","쫄면");

        //응답할 menus를 JSON객체로 변환해서 응답
        Gson gson = new Gson();
        String jsonArry = gson.toJson(menus);
        System.out.println("jsonArray = " + jsonArry);
       //응답
        PrintWriter out = response.getWriter();
        out.println(jsonArry);
    }
```

-DTO결과

```
// DTO 객체
document.getElementById("dtoBtn").addEventListener("click", async function () {
    try {
        const response = await fetch(`${"${baseURL}"}/ajax?key=dto`);
        if (!response.ok) throw new Error("DTO 요청 실패");

        const result = await response.json();
        console.log(result);

       // const str = `${result.id} ¦ ${result.name} ¦ ${result.addr} ¦ ${result.age}`;
       const str = result.id + " ¦ "  + result.name + " ¦ " + result.addr +  " ¦ " +  result.age;
        document.getElementById("display").innerHTML = str;
    } catch (error) {
        console.log("에러:", error);
    }
});
```

8253jang@daum.net
장희정

```java
    /**
     * DTO 결과
     * */
    public void dto(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        Member member =
          new Member("jang", "장희정", 25, "서울");

         Gson gson = new Gson();
         String jsonArr = gson.toJson(member);
          //응답
         PrintWriter out = response.getWriter();
         out.println(jsonArr);

    }
```

List결과

```javascript
// List
document.getElementById("listBtn").addEventListener("click", async function () {
    try {
        const response = await fetch(`${"${baseURL}"}/ajax?key=list`);
        if (!response.ok) throw new Error("List 요청 실패");

        const result = await response.json();
        console.log(result);
    } catch (error) {
        console.log("에러:", error);
    }
});

```

```java
    /**
     * List결과
     * */
    public void list(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        List<Member> list = new ArrayList<Member>();
        list.add(new Member("jang", "장희정", 25, "서울"));
        list.add(new Member("kim", "이흘", 30, "서울"));
        list.add(new Member("king", "이나영", 27, "대구"));
        list.add(new Member("aaa", "김희선", 25, "부산"));
        list.add(new Member("test", "장동건", 20, "대전"));

         Gson gson = new Gson();
         String jsonArr = gson.toJson(list);
         System.out.println("jsonArr = " + jsonArr);
          //응답
         PrintWriter out = response.getWriter();
         out.println(jsonArr);

    }
```

8253jang@daum.net
장희정

Map결과

```
// Map (ex: Map<String, Object> → {"memberList":[...]})
document.getElementById("mapBtn").addEventListener("click", async function () {
    try {
        const response = await fetch(`${"${baseURL}"}/ajax?key=map`);
        if (!response.ok) throw new Error("Map 요청 실패");

        const result = await response.json();
        console.log(result);

        result.memberList.forEach(member => {
            console.log(member.name);
        });
    } catch (error) {
        console.log("에러:", error);
    }
});
```

```
 /**
  * Map결과
  * */
public void map(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    Map<String, Object> map = new HashMap<>();
    map.put("message", "Ajax재미있다.");
    map.put("pageNo", 20);
    map.put("dto", new Member("jang", "장희정", 25, "서울"));


    List<Member> list = new ArrayList<Member>();
    list.add(new Member("jang", "장희정", 25, "서울"));
    list.add(new Member("kim", "이효", 30, "서울"));
    list.add(new Member("king", "이나영", 27, "대구"));
    map.put("memberList", list);


     Gson gson = new Gson();
     String jsonArr = gson.toJson(map);
     System.out.println("jsonArr = " + jsonArr);
      //응답
     PrintWriter out = response.getWriter();
     out.println(jsonArr);
 }
```

8253jang@daum.net
장희정

TodoList 실습하기

기능	Method	Path	Body	성공(Response)
전체검색	POST	/ajax	{ "key": "todoList" , "methodName": "selectAll" }	[{"id":10,"done":false,"content":"학교 다녀오기","date":"2025-03-31 08:29:17"} ,{"id":11,"done":false,"content":"엄마랑 통화하기","date":"이 PC에 저장됨 9:23"}, {"id":12,"done":false,"content":"숙제하기","date":"2025-03-31 08:29:27"}]
등록	POST	/ajax	{ "key": "todoList" ,"methodName": "insert", "content": "..." }	0 or 1
수정	POST	/ajax	{ "key": "todoList" ,"methodName": "update", "id": 10, "done": true }	0 or 1
삭제	POST	/ajax	{"key": "todoList" , "methodName": "delete", "targetId": 10 }	0 or 1
키워드검색	POST	/ajax	{ "key": "todoList" ,"methodName": "selectIncludesByWord", "word": "..." }	[{"id":10,"done":false,"content":"학교 다녀오기","date":"2025-03-31 08:29:17"} {"id":12,"done":false,"content":"숙제하기","date":"2025-03-31 08:29:27"}]