02

JavaScript와 ECMAScript

학습내용

01 ECMAScript 버전별 주요 특징

02 JavaScript의 성장 역사

03 JavaScript의 특징

학습목표

- ECMAScript 1부터 최신 버전까지의 핵심 변경 사항을 이해하고, 각 버전에서 도입된 주요 문법과 기능의 의미를 설명할 수 있다.
- JavaScript의 초기 도입 배경부터 현재까지의 발전사를 시대별로 설명할 수 있다.
- 브라우저 기술과 개발 생태계 변화에 따라 JavaScript가 어떻게 진화했는지 설명할 수 있다.

01

ECMAScript 버전별 주요 특징

1) ECMAScript 버전별 주요 특징 요약

ES6부터는 연도 기반으로 명명되기 시작함

예 ES6 = ES2015, ES7 = ES2016…

버전	출시연도	주요 특징 요약
ES1	1997년	• 최초의 공식 표준 • 기본 문법 및 연산자, 제어문 등 정의
ES2	1998년	• ISO 표준에 맞춘 사소한 문서 정리 및 포맷 변경
ES3	1999년	• 중대한 기능 추가 – 정규표현식, try-catch, do-while, switch 등
ES4	(취소됨)	• 기능 과잉과 브라우저 간 합의 실패로 공식 출시 무산

1) ECMAScript 버전별 주요 특징 요약

ES6부터는 연도 기반으로 명명되기 시작함

예 ES6 = ES2015, ES7 = ES2016…

버전	출시연도	주요 특징 요약
ES5	2009년	• **현대 JS 기반** – strict mode, JSON 지원, Array 메서드 개선(forEach, map 등), 객체 프로퍼티 제어(Object.defineProperty)
ES6 (ES2015)	2015년	• **대형 업데이트** – let/const, 화살표 함수, 클래스, 모듈, Promise, Map/Set, 템플릿 리터럴 등
ES7 (ES2016)	2016년	• Array.prototype.includes, ** (거듭제곱 연산자) 도입

1) ECMAScript 버전별 주요 특징 요약

ES6부터는 연도 기반으로 명명되기 시작함

예 ES6 = ES2015, ES7 = ES2016…

버전	출시연도	주요 특징 요약
ES8 (ES2017)	2017년	• async/await, Object.entries(), Object.values(), 문자열 패딩
ES9 (ES2018)	2018년	• rest/spread for objects, Promise.finally(), 비동기 반복문(for await...of)
ES10 (ES2019)	2019년	• flat(), flatMap(), trimStart()/trimEnd(), Object.fromEntries() 등
ES11 (ES2020)	2020년	• BigInt, nullish coalescing, 옵셔널 체이닝 연산wk, Promise.allSettled(), globalThis

“ES3, ES5, ES6가 JavaScript 발전의 큰 전환점이 된 버전”

버전	핵심 의미	주요 기능 키워드
ES3	JavaScript의 최초 초안 완성	• try/catch 예외 처리 도입, do...while, switch, for-in 등의 제어문 강화, delete, typeof, instanceof 같은 핵심 연산자 도입 • 브라우저 간 호환성 중심으로 표준화
ES5	브라우저 호환성, 코드의 안정성과 구조화 강화	• strict mode(“use strict”): 오류 예방, 암묵적 전역변수 차단 • Object.defineProperty : 객체 속성 제어 • Array 메서드 추가 : forEach, map, filter, reduce 등 • JSON 지원 내장, getter/setter 도입
ES6	모던 JavaScript의 시작 (프로그래밍 수준향상)	• Arrow Function(=>) • Class 문법(OOP 스타일) • Module(import / export) • Promise(비동기 처리의 표준화)

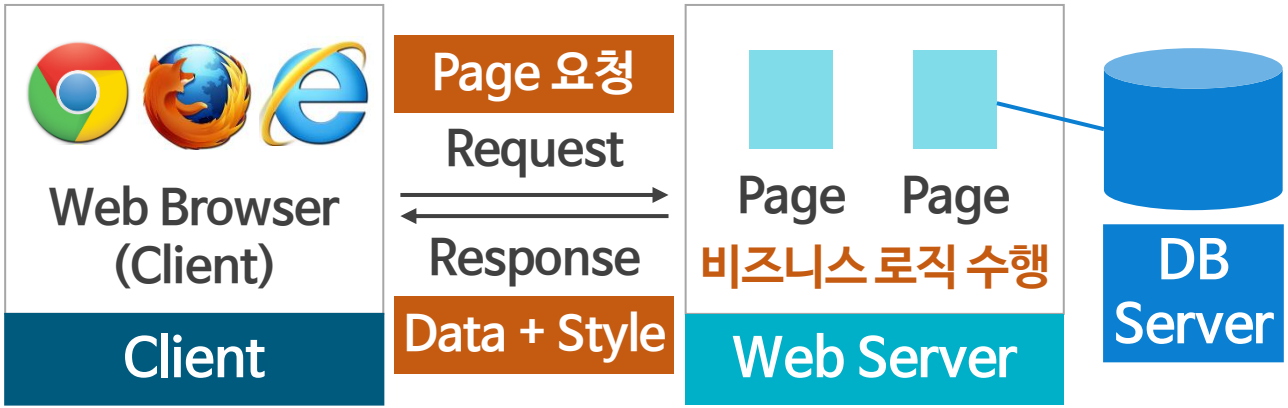
02

JavaScript의 성장 역사

1) 초창기

- 웹 페이지의 보조적인 기능을 수행하기 위해 한정적인 용도로 사용

- 이 시기에 대부분의 로직은 주로 웹 서버에서 실행되고, 브라우저는 서버로부터 전달받은 HTML과 CSS를 단순히 렌더링하는 수준



JavaScript를 이용해
서버와 브라우저가 비동기(Asynchronous)방식으로
데이터를 교환할 수 있는 통신 기능

XMLHttpRequest라는 이름으로 등장

출처 위키피디아

이전의 웹 페이지

- HTML 태그로 시작해서 HTML 태그로 끝나는 완전한 HTML 코드를 서버에서 전송받아 웹 페이지 전체를 렌더링하는 방식으로 동작

참고 **렌더링**

명령어를 해석해 브라우저에 시각적으로 출력하는 것

- 화면이 전환되면 서버로부터 새로운 HTML을 전송받아 웹 페이지 전체를 처음부터 다시 렌더링

불필요한 데이터 통신 발생, 성능 불리 → 화면이 전환되면 화면이 순간적으로 깜빡이는 현상

AJAX의 등장

서버로부터 필요한 데이터만 전송 받아 변경해야 하는 부분만
한정적으로 렌더링하는 방식이 가능함

웹 브라우저에서도 데스크톱 애플리케이션과 유사한
빠른 성능과 부드러운 화면 전환이 가능해짐

출처 위키피디아

다소 번거롭고 논란이 있던 DOM을 더욱 쉽게 제어할 수 있게 됨

크로스 브라우징 이슈도 어느 정도 해결됨

“ V8 JavaScript 엔진 ”

프로그래밍 언어로서 가능성이 확인된 JavaScript로
웹 애플리케이션을 구축하려는 시도 증가

↓

더욱 빠르게 동작하는 JavaScript 엔진의 필요성 대두

↓

2008년에 등장한 구글의 V8 JavaScript 엔진은
이러한 요구에 부합

V8 JavaScript 엔진의 등장

- 과거 웹 서버에서 수행되던 로직들이 대거 클라이언트(브라우저)로 이동했고, 웹 애플리케이션 개발에서 프론트엔드 영역이 주목받는 계기로 작용함

- JavaScript는 데스크톱 애플리케이션과 유사한 사용자 경험(UX)을 제공할 수 있는 웹 애플리케이션 프로그래밍 언어로 정착하게 됨

5) Node.js

출처 위키피디아

- 구글 V8 JavaScript 엔진으로 빌드된 JavaScript 런타임 환경
- 브라우저 이외의 환경에서도 동작할 수 있도록 JavaScript 엔진을 브라우저에서 독립시킨 JavaScript 실행 환경

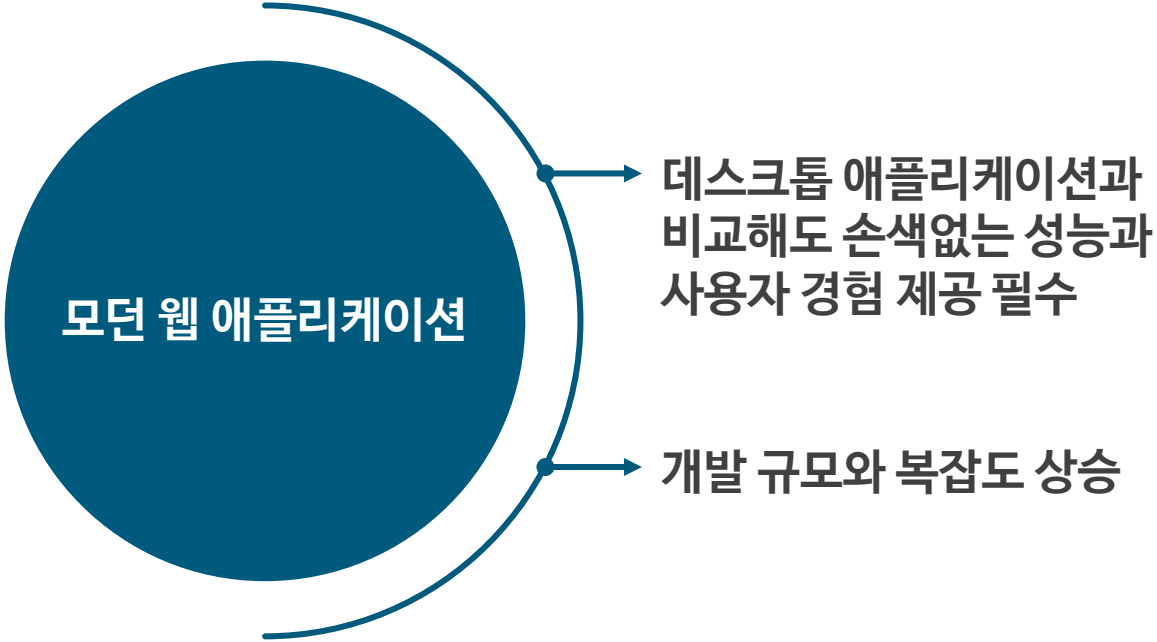
5) Node.js

- **서버 사이드 애플리케이션 개발에 주로 사용**하며, 이에 필요한 내장 API를 제공함
- 비동기 I/O를 지원하며 단일 스레드 이벤트 루프 기반으로 동작함으로써 요청(Request) 처리 성능이 좋음
  - ➡ 데이터를 실시간으로 처리하기 위해 I/O가 빈번하게 발생하는 SPA(Single Page Application)에 적합
- CPU 사용률이 높은 애플리케이션에서는 권장하지 않음
- Node.js 등장으로 JavaScript는 범용 프로그래밍 언어가 됨

> 이제 JavaScript는 크로스 플랫폼을 위한 가장 중요한 언어로 주목받고 있음

웹, 모바일 하이브리드 앱	PhoneGap, Ionic
서버 사이드	Node.js
데스크톱	Electron
머신러닝	TensorFlow.js
로보틱스	Johnny-Five

6) SPA 프레임워크



"많은 패턴과 라이브러리 ➡ 프레임워크 등장"

6) SPA 프레임워크

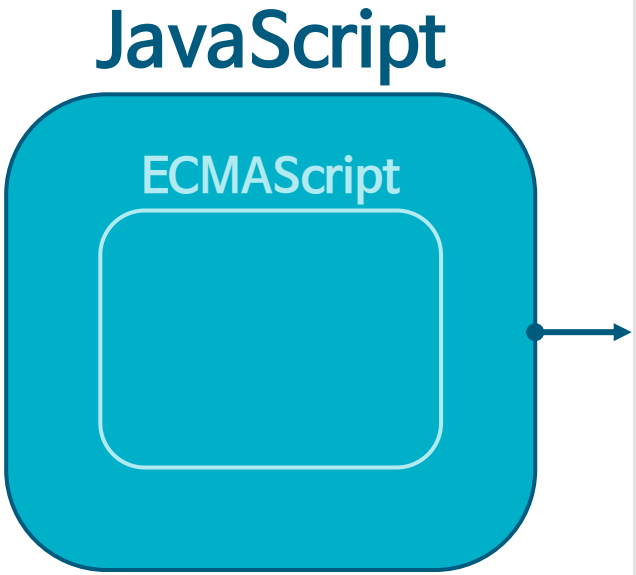
CBD(Component based development) 방법론을 기반으로 하는 SPA(Single Page Application)가 대중화가 되면서 Angular, React, Vue.js, Svelte 등 다양한 SPA 프레임워크/라이브러리 사용

참고 **용어 정리**

- MPA(Multi Page Application)
  - 페이지마다 각각의 HTML 파일을 갖는 전통적인 웹 애플리케이션 구조
- SPA(Single Page Application)
  - 웹 애플리케이션의 한 형태로, 페이지 전체를 매번 새로고침하지 않고, 필요한 부분만 동적으로 바꿔서 보여주는 방식

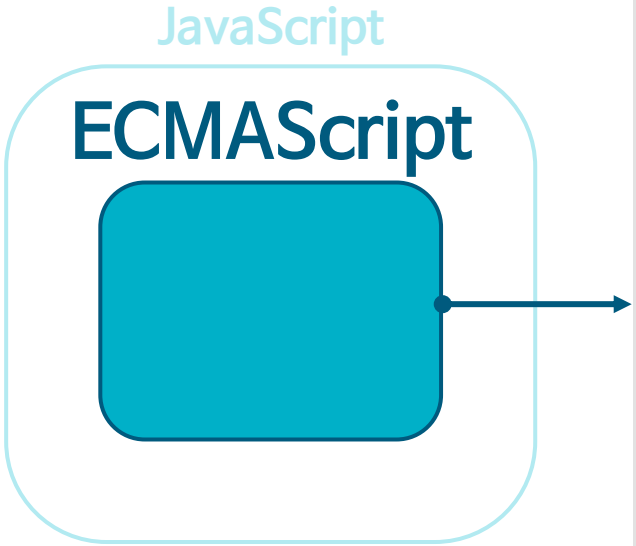
03

JavaScript의 특징



- 웹 브라우저에서 실행되는 프로그래밍 언어임
- 웹 페이지를 동적으로 만들고 사용자와 상호작용할 수 있게 함

2) ECMAScript



- JavaScript 표준 사양인 ECMA-262를 뜻함
- 프로그래밍 언어의 값, 타입, 객체와 프로퍼티, 함수, 표준 빌트인(Built-in) 객체 등 핵심 문법 규정임
- 각 브라우저 제조사는 ECMAScript 사양을 준수해서 브라우저에 내장되는 JavaScript 엔진을 구현함
- 일반적으로 프로그래밍 언어로서 기본 뼈대(Core)를 이루는 것임

3) 클라이언트 사이드 Web API

- DOM
- BOM
- Canvas
- XMLHttpRequest
- fetch
- requestAnimationFrame
- SVG
- Web Storage
- Web Component
- Web Worker 등

4) JavaScript의 특징

- 웹 브라우저에서 동작하는 유일한 프로그래밍 언어

- 개발자가 별도의 컴파일 작업을 수행하지 않는 **인터프리터 언어**

  - 대부분의 모던 JavaScript 엔진은 인터프리터와 컴파일러의 장점을 결합해 비교적 처리 속도가 느린 인터프리터의 단점 해결
    - 컴파일러 언어처럼 명시적인 컴파일 단계를 거치지는 않지만 복잡한 과정을 거치며 일부 소스코드를 컴파일하고 실행함
      - ➡ 인터프린터 언어의 장점인 동적 기능 지원 살리면서 실행속도 느리다는 단점 극복
    - 런타임에 컴파일되며 실행 파일이 생성되지 않고 인터프리터의 도움 없이 실행할 수 없기 때문에 컴파일 언어라 할 수 없음

컴파일러 언어

소스코드 전체를 한번에 머신 코드로 변환한 후 실행함

실행 파일을 생성함

컴파일 단계와 실행 단계가 분리되어 있음

실행에 앞서 컴파일은 단 한 번 수행됨

컴파일과 실행 단계가 분리되어 있으므로 코드 실행 속도가 빠름

인터프리터 언어

한 줄씩 중간코드인 바이트코드로 변환한 후 실행함

실행 파일을 생성하지 않음

인터프리트 단계와 실행 단계가 분리되어 있지 않음

코드가 실행될 때마다 인터프리트 과정이 반복 수행됨

인터프리트 단계와 실행 단계가 분리되어 있지 않고 반복 수행하므로 코드 실행 속도가 비교적 느림

4) JavaScript의 특징

- 명령형, 함수형, 프로토타입 기반 객체지향 프로그래밍을 지원하는 멀티 패러다임 프로그래밍 언어

- 클래스 기반이 아닌 프로토타입 기반의 객체지향 언어

참고 **프로토타입이란?**

객체 간의 상속이 아닌
다른 객체를 참조하는 방식

세종사이버대학교

01주. JavaScript의 개요

03

JavaScript의 실행 환경 및 디버깅