📘 04. 구조와 API를 왜 분리해서 이해해야 할까?

1. DOM의 본질을 다시 바라보며

지금까지 우리는 DOM이라는 것이 무엇인지, 왜 필요한지, 그리고 HTML이 어떻게 DOM으로 바뀌는지에 대해 하나씩 배워왔습니다.

→ DOM은 브라우저가 HTML을 파싱해 생성한 메모리상의 구조이며, 자바스크립트는 이 DOM을 통해 웹페이지를 조작할 수 있습니다.

이제부터는 시야를 조금 더 넓혀서, DOM을 구성하는 두 가지 핵심 요소—구조(structure)와 **API(interface)**—를 명확히 구분해서 이해하는 연습을 시작해보겠습니다.

→ 'DOM이란 무엇인가?'라는 질문에 대해 이제는 단일한 답이 아니라, **구조(트리)** 와 **조작 도구(API)** 를 나눠서 바라보는 시야가 필요합니다.

이 두 개념을 분리하지 않고 혼동한 채로 DOM을 다루다 보면, 나중에 DOM을 조작하거나 에러가 발생했을 때 **문제의 원인을 파악하기 어려워질 수 있습니다.**

→ DOM 구조 자체의 오류인지, API 호출 방식의 문제인지 구분하지 못하면 디버깅이 매우 어려워집니다.

따라서 이 시점에서 구조와 API를 정확히 나누어 정리해두는 것이 매우 중요합니다.

2. 비유로 이해하기: DOM 구조는 건물, DOM API는 공구

DOM을 잘 이해하기 위해서는 먼저 DOM이 어떤 '모습'을 갖고 있는지, 그리고 우리가 어떤 '방법'으로 그것을 다룰 수 있는지를 따로 구분해서 보는 연습이 필요합니다.

이를 이해하는 좋은 비유는 바로 **건물과 공구**입니다.

→ 건물: HTML을 해석해 만들어진 DOM 구조

→ 공구: DOM을 조작하기 위한 API 함수들

건물은 설계도에 따라 지어진 구조물이고, 드릴이나 망치 같은 공구는 그 건물을 유지보수하거나, 내부를 바꾸거나, 확장하기 위한 도구입니다.

DOM의 구조는 바로 그 건물입니다. 우리가 HTML로 작성한 코드가 브라우저에 의해 DOM 트리라는 건물로 조립된 것이죠.

→ DOM 트리는 태그 간의 중첩 구조를 반영하여, 부모-자식 노드로 연결된 객체 트리입니다.

반면, 자바스크립트로 그 DOM을 다룰 때 사용하는 다양한 메서드들—예를 들어 `getElementById`, `appendChild`, `setAttribute` 같은 것들은 공구입니다.

DOM API는 바로 이 공구 상자라고 생각하면 됩니다.

→ 이 API는 브라우저가 자바스크립트에게 제공하는 **표준화된 조작 도구 모음집**입니다.

---

3. 구조와 도구를 동시에 살펴보는 실습 예시

좀 더 구체적으로 설명해보겠습니다. 우리가 아래와 같은 HTML을 작성했다고 해봅시다.

```
<body>
  <h1 id="title">처음 제목</h1>
  <button>제목 바꾸기</button>
</body>
```

이 코드를 브라우저가 해석하면, HTML은 DOM 트리로 변환됩니다.

→ 브라우저는 `<body>`를 루트로 삼고, 그 아래에 `<h1>`과 `<button>`이라는 두 개의 요소 노드를 추가하여 **트리 구조**를 형성합니다.

`<body>`는 최상위 노드가 되고, 그 안에 `<h1>`과 `<button>`이라는 두 개의 자식 노드가 존재하는 구조가 메모리 속에 만들어지죠.

여기까지는 **구조**입니다. 우리가 코드를 작성한 순간 자동으로 생긴, 조작되기 전의 기본적인 형태입니다.

→ 이 단계에서는 아직 아무런 상호작용도 정의되지 않았습니다.

---

4. DOM 구조 위에서 DOM API를 사용해 조작하기

하지만 이 페이지에서 “제목 바꾸기” 버튼을 눌렀을 때 `<h1>`의 텍스트가 바뀌기를 원한다고 가정해봅시다.

그러면 자바스크립트를 이용해 이 DOM 구조에 접근하고 조작해야 합니다.

→ 이때 사용하는 것이 바로 **DOM API**입니다.

예를 들어 다음과 같은 자바스크립트 코드를 작성하면 됩니다:

```
const h1 = document.getElementById("title");
const button = document.querySelector("button");

button.addEventListener("click", () => {
  h1.textContent = "바뀐 제목입니다!";
});
```

📌 코드 설명:

- `getElementById("title")`

  → id가 "title"인 요소를 DOM 트리에서 탐색하여 반환 (DOM API)

- `querySelector("button")`

  → CSS 선택자 기반으로 `<button>` 요소를 선택 (DOM API)

- `addEventListener("click", ...)`

  → 이벤트 리스너를 등록하여 클릭 시 특정 함수를 실행

- `h1.textContent = ...`

  → 노드의 텍스트 콘텐츠를 변경 (DOM 조작)

이로써 실제 웹페이지 화면에 보이는 `<h1>`의 텍스트가 “처음 제목”에서 “바뀐 제목입니다!”로 바뀌게 됩니다.

---

5. 구조는 고정되고, API는 작동한다

이 예시의 핵심은 **DOM 구조는 미리 존재하고, DOM API는 그 위에서 일어나는 일**이라는 점입니다.

→ 자바스크립트는 DOM을 생성하지 않으며, **이미 브라우저가 만든 구조를 탐색하고 수정**할 수만 있습니다.

구조는 건물이고, API는 도구입니다. 구조는 변하지 않지만, 도구를 사용해서 그 구조의 일부분을 교체하거나 변경할 수 있는 것이죠.

좀 더 쉽게 설명하면, DOM 구조는 마치 집의 벽, 천장, 창문처럼 원래부터 존재하는 뼈대이고, DOM API는 그 집에서 전구를 교체하거나, 벽에 그림을 거는 행위라고 생각하면 됩니다.

→ HTML을 기반으로 브라우저가 지은 구조물은 변화 없이 존재하고, 그 위에서 자바스크립트는 **DOM API라는 도구로 특정 부분을 제어**합니다.

---

6. DOM API는 브라우저가 제공하는 공식 도구 모음집이다

즉, 자바스크립트를 통해 DOM을 조작한다는 건, 이미 만들어진 구조 안에서 어떤 것의 상태나 내용을 바꾸는 일입니다.

그리고 그 조작을 하기 위해 제공되는 공식적인 도구들이 바로 DOM API입니다.

DOM API는 쉽게 말해 **브라우저가 우리에게 제공해주는 도구 모음집입니다.**

전등을 켜려면 스위치를 눌러야 하듯, DOM을 바꾸려면 이 API를 '호출'해야만 합니다.

우리가 직접 만든 함수가 아니라, 브라우저가 "DOM 구조를 이렇게 다룰 수 있어요"라고 미리 정해놓은 **공식적인 함수들**입니다.

예를 들어 `getElementById()`, `appendChild()`, `setAttribute()` 같은 메서드들은 우리가 이름을 바꿀 수 없고, 모든 브라우저에서 동일하게 작동해야만 합니다.

이것이 바로 **표준 DOM API**입니다.

→ 이러한 API는 DOM 명세서에 정의된 대로 동작해야 하며, 웹 표준을 따르는 모든 브라우저에서 동일한 결과를 보장합니다.

---

7. 구조와 API의 분리 이해가 중요한 이유

이 개념이 중요한 이유는, 우리가 앞으로 배워갈 수많은 DOM 관련 기능들—예를 들어 요소 선택자, 노드 탐색, 속성 변경, 스타일 조작, 이벤트 연결—이 모두 **DOM 구조 위에서 DOM API를 사용해 일어나는 일**이기 때문입니다.

만약 DOM이 건물처럼 존재하지 않는다면, 그 위에서 어떤 도구도 사용할 수 없을 겁니다.

반대로 DOM API를 몰라서 조작을 못 한다면, 건물은 있어도 안을 고칠 수 없는 상태가 되는 것이죠.

결국 진짜로 DOM을 다룬다는 것은, 구조와 도구—즉, DOM 트리와 DOM API—두 가지를 모두 이해하고 활용할 수 있어야 가능하다는 뜻입니다.

✅ 핵심 요약

정리하자면, DOM을 이해할 때는 **두 가지 시야를 동시에 가져야 합니다.**

하나는 "DOM 트리 구조를 이해하는 시야"이고,

다른 하나는 "그 구조를 다루는 방법, 즉 API를 아는 시야"입니다.

이 둘이 함께 있어야 진짜 DOM을 자유자재로 다룰 수 있는 개발자가 됩니다.

→ 구조를 모르면 조작 대상이 없고, API를 모르면 조작 자체가 불가능합니다.
