22. 📘 DOM 완전 정복 Part 1 – 22강. DOM 요소 제거와 교체: remove, replaceChild, innerHTML, textContent 완벽 정리

1. 단순 추가만으로는 부족하다

지금까지 우리는 HTML 요소를 새로 만들고 원하는 위치에 삽입하는 다양한 방법들을 배워왔습니다. 하지만 실전 웹페이지에서는 단순히 요소를 추가하는 것만으로는 부족합니다. 어떤 요소는 사라져야 하고, 어떤 요소는 완전히 새로운 것으로 **교체되어야** 합니다.

예를 들어,

- 댓글 삭제 버튼을 누를 때 → 댓글 DOM 제거
- 장바구니에서 상품 제거 → 특정 `<li>` 제거
- 경고 메시지 닫기 → 알림창 제거

이 모든 상황은 DOM에서 **노드를 제거하거나 교체하는 과정**입니다.

이번 시간에는 DOM 조작에서 꼭 알아야 할 **요소 제거 및 교체 메서드들**을 정리하고, 각 방식의 차이점, 내부 동작 방식, 그리고 보안 및 성능 측면까지 함께 살펴보겠습니다.

---

2. `remove()` – 가장 간단한 제거 방식

🔧 개념 설명

`.remove()`는 가장 직관적인 방식의 DOM 제거 메서드입니다. **선택된 요소를 DOM에서 바로 제거**합니다.

💡 코드 예제

```
<ul id="fruits">
  <li>🍎 Apple</li>
  <li id="banana">🍌 Banana</li>
```

```
  <li>🍇 Grape</li>
</ul>
```

```
const banana = document.getElementById("banana");
banana.remove();
```

→ 실행 결과: `🍌 Banana`가 화면에서 사라짐

🧠 실무 팁

- 이벤트 리스너 안에서 특정 DOM 요소를 직접 삭제할 때 자주 사용
- 최신 브라우저(IE 미지원)에서만 사용 가능

---

3. `removeChild()` – 브라우저 호환성 높은 전통 방식

🔧 개념 설명

`removeChild()`는 **부모 노드에서 자식 노드를 제거**하는 방식입니다.

```
banana.parentNode.removeChild(banana);
```

→ `banana`의 부모(`ul`)가 직접 자식을 제거함

💡 내부 동작

`banana.remove()`는 사실상 다음과 동일하게 작동:

```
this.parentNode.removeChild(this);
```

→ `remove()`는 내부적으로 부모를 참조해 자기 자신을 제거하는 **단축형 메서드**

⚠️ 차이점

- `remove()`는 부모가 없어도 조용히 실패
- `removeChild()`는 부모가 없거나 자식이 아닐 경우 → **오류 발생**

---

4. `replaceChild()` – 기존 노드를 새로운 노드로 교체

🔧 개념 설명

`replaceChild(newNode, oldNode)`는 말 그대로 **자식 노드를 새 노드로 교체**합니다.

```
parentNode.replaceChild(newNode, oldNode);
```

💡 코드 예제

```
const list = document.getElementById("fruits");
const newItem = document.createElement("li");
newItem.textContent = "🍊 Orange";

const banana = document.getElementById("banana");
list.replaceChild(newItem, banana);
```

→ 실행 결과: Banana가 Orange로 바뀜

🧠 실무 팁

- 실시간 데이터 업데이트 시(주식, 점수판 등) 전체 영역을 **새 노드로 통째로 교체**
- `replaceChild()`는 사실상 → 기존 노드 삭제 + 새 노드 삽입의 묶음 단축 API

---

5. `innerHTML` vs `textContent` – 내부 내용 수정의 두 방식

📌 `innerHTML`

→ 문자열을 **HTML로 해석**해서 DOM을 재구성

```
box.innerHTML = "<strong>Hello!</strong>";
```

→ `<strong>`이 실제로 적용됨

- 장점: HTML 구조 포함 가능
- 단점: XSS 위험, 리렌더링 비용 큼

📌 textContent

→ 문자열을 **텍스트 그대로 삽입**

```
box.textContent = "<strong>Hello!</strong>";
```

→ `<strong>Hello!</strong>`가 문자열로 출력됨

- 장점: 빠르고 안전함 (XSS 차단)
- 단점: HTML 불가능

---

6. 보안과 성능 측면의 핵심 차이

속성	XSS 위험	HTML 삽입 가능	성능
innerHTML	높음	O	느림 (DOM 재구성)
textContent	없음	X	빠름 (단순 텍스트 교체)

> 🛡️ 실무 원칙: 사용자 입력값은 반드시 textContent로 출력!

✅ 핵심 요약

- `.remove()` → 자기 자신을 직접 제거 (최신 브라우저 전용)
- `removeChild()` → 부모가 자식을 제거 (호환성 높음)
- `replaceChild(new, old)` → old를 new로 교체
- `innerHTML` → HTML 구조를 넣거나 가져옴 (XSS 주의)
- `textContent` → 텍스트만 안전하게 다룸
