20. 📘 요소를 만들고 추가하는 방법 – appendChild(), prepend()

1. 정적인 HTML을 넘어서 – DOM에 요소를 직접 추가하는 기술

지금까지 우리는 이미 존재하는 DOM 요소를 선택하고 탐색하는 방법에 집중해왔습니다. 하지만 진짜 웹페이지는 단순히 정적인 요소를 보여주는 것에 그치지 않고, 사용자의 행동이나 서버의 응답에 따라 **새로운 요소를 직접 만들어 추가하는 작업**이 필수적으로 요구됩니다.

→ 예: 댓글 작성 시 새로운 댓글이 즉시 추가되거나

→ 쇼핑몰에서 '장바구니에 담기' 버튼을 누르면 해당 상품이 리스트에 추가되는 동적 UI

이제부터는 DOM에 새 요소를 만드는 법, 추가하는 법, 그리고 효율적으로 대량의 요소를 처리하는 방법까지 하나씩 배워봅시다.

---

2. `createElement()` 로 새로운 요소 만들기

자바스크립트로 새로운 DOM 요소를 만들려면 `document.createElement("태그명")`을 사용합니다.

```
const newItem = document.createElement("li");
newItem.textContent = "Contact";
```

→ `"li"` 태그 요소를 새로 생성하고, 텍스트 내용을 “Contact”로 지정한 상태입니다.

→ 아직 DOM에는 추가되지 않은, 메모리에만 존재하는 상태입니다.

---

3. `appendChild()` 로 마지막에 자식 추가

만든 요소는 화면에 보이게 하려면 DOM에 **명시적으로 연결**해야 합니다. 가장 기본적인 연결 방식은 `appendChild()`입니다.

```
<ul id="menu">
  <li>Home</li>
  <li>About</li>
</ul>
```

```
const menu = document.getElementById("menu");
menu.appendChild(newItem);
```

→ `appendChild()`는 부모 요소의 **마지막 자식 위치**에 새 요소를 추가합니다.

→ 실제 DOM 구조는 이렇게 바뀝니다:

```
<ul id="menu">
  <li>Home</li>
  <li>About</li>
  <li>Contact</li> <!-- 추가된 항목 -->
</ul>
```

---

4. `prepend()`로 가장 앞에 자식 추가

`prepend()`는 자식 요소를 **맨 앞**에 추가합니다. 구조는 그대로인데 위치만 다릅니다.

```
menu.prepend(newItem);
```

→ 이 경우 "Contact"가 가장 첫 번째 항목으로 들어갑니다.

html

```
<ul id="menu">
  <li>Contact</li> <!-- 가장 앞에 추가됨 -->
  <li>Home</li>
  <li>About</li>
</ul>
```

📌 `appendChild()`는 마지막에, `prepend()`는 맨 앞에 추가합니다.

---

5. 실제 사용자 시나리오 예시

✔️ 댓글 최신순 → `prepend()`

✔️ 댓글 오래된 순 → `appendChild()`

✔️ 실시간 채팅 메시지 출력 → 둘 다 사용 가능 (위에서부터 or 아래에서부터)

✔️ 알림 리스트, 뉴스 피드 → 상황에 따라 유동적으로 사용

→ 화면에 보이는 위치와 사용자의 경험(UX)을 고려하여 선택해야 합니다.

---

6. 성능이 중요한 순간 – 대량 추가는 `DocumentFragment` 로!

`appendChild()`를 반복문 안에서 수백 번 호출하면 렌더링이 그때마다 일어나 성능이 급격히 저하될 수 있습니다. 이럴 땐 `DocumentFragment`라는 **보이지 않는 가상의 DOM**에 먼저 요소들을 모아두었다가 **한 번에 붙이는 방법**이 효과적입니다.

```
const menu = document.getElementById("menu");
const fragment = document.createDocumentFragment();

for (let i = 0; i < 1000; i++) {
  const li = document.createElement("li");
  li.textContent = `Item ${i + 1}`;
  fragment.appendChild(li);
}

menu.appendChild(fragment); // 한 번만 렌더링 발생
```

→ 이 방식은 **브라우저의 reflow/repaint를 최소화**하여 성능을 대폭 향상시킵니다.

---

7. 일상적 비유 – 왜 `DocumentFragment` 가 빠를까?

마치 마트에서 물건을 한 개씩 카트에 넣으며 계산대에 가는 것보다,

박스에 미리 담아서 한 번에 가는 것이 빠른 것과 비슷합니다.

→ 매번 `appendChild()`를 하면 그때마다 계산대(reflow)가 동작

→ `DocumentFragment`는 **한 번의 계산**으로 모든 요소를 반영

---

8. 브라우저 동작과 성능 관점 – `reflow` , `repaint` , `layout`

DOM에 요소를 추가하면, 브라우저는 다음과 같은 일련의 작업을 처리합니다:

→ DOM 구조 변경 (reflow)

→ 스타일 재계산 (CSSOM update)

→ 실제 화면에 그리기 (repaint)

이 작업이 반복문 안에서 너무 자주 일어나면 CPU와 메모리 사용량이 급증하고

→ 화면이 깜빡이거나,

→ 입력이 지연되거나,

→ 모바일에서 스크롤이 끊기는 현상이 발생합니다.

`DocumentFragment`는 이 문제를 우아하게 해결하는 훌륭한 방법입니다.

---

9. 실습 예제 – 댓글 1000개 추가

```
const commentBox = document.getElementById("comments");
const fragment = document.createDocumentFragment();

for (let i = 0; i < 1000; i++) {
  const div = document.createElement("div");
  div.className = "comment";
  div.textContent = `댓글 ${i + 1}`;
  fragment.appendChild(div);
}

commentBox.appendChild(fragment);
```

→ 실전에서도 데이터 리스트, 피드 추가 등에서 매우 자주 사용됩니다.

✅ 핵심 요약

- `createElement("태그명")` → 새로운 요소 생성 (아직 화면에 없음)
- `appendChild()` → 부모 요소의 마지막에 자식 추가
- `prepend()` → 부모 요소의 맨 앞에 자식 추가
- `DocumentFragment` → 가상의 DOM 공간, 반복 추가 시 성능 최적화에 필수
- 성능 저하를 방지하려면 반복 렌더링을 피하고 **한 번에 추가**하는 방식 사용
- 브라우저는 요소 추가 시 `reflow`, `repaint`, `layout`을 수행하므로 신중한 구조 설계 필요
