16. 📘 getElementsByTagName()의 진짜 모습 – live DOM 컬렉션과 안전한 순회 전략

1. 특정 태그를 한꺼번에 선택하는 가장 기본적인 방법

`getElementsByTagName()` 메서드는 HTML 문서 안에 있는 특정 태그들을 한꺼번에 선택할 때 자주 사용되며, 그 결과로 `HTMLCollection`이라는 특별한 자료 구조를 반환합니다.

이전 강의에서는 `HTMLCollection`과 `NodeList`가 겉보기에는 유사해 보이지만, 실제로는 매우 다른 동작을 한다는 점을 확인했습니다.

특히 `HTMLCollection`은 DOM의 변화에 실시간으로 반응하는 **live 구조**이고, `NodeList`는 한 번 선택된 상태를 유지하는 **정적인 구조**입니다.

이번 강의에서는 `getElementsByTagName()`이 **HTMLCollection을 반환하는 대표적인 메서드**로서 실제로 어떻게 작동하는지, 또 어떤 주의사항이 있는지를 예제 중심으로 알아보겠습니다.

---

2. 기본 사용법 – 특정 태그명을 기준으로 요소 선택

```
const listItems = document.getElementsByTagName("li");
```

→ 이 코드는 문서 안에 있는 모든 `<li>` 태그를 찾아 `listItems`라는 이름의 HTMLCollection으로 반환합니다.

→ 이 구조는 마치 배열처럼 생겼지만 실제 배열은 아닙니다. 가장 중요한 특징은 이 컬렉션이 **live 상태**라는 점입니다.

→ 예를 들어 새로운 `<li>` 요소가 DOM에 추가되면 `listItems.length`는 자동으로 증가하게 됩니다.

---

3. 가져온 항목의 내용을 수정해보기

```
listItems[0].textContent = "첫 번째 항목 변경!";
```

→ 첫 번째 `<li>` 요소의 텍스트가 즉시 바뀝니다.

→ HTMLCollection은 배열처럼 인덱스를 통해 접근할 수 있고, `.textContent`나 `.style` 등을 사용해 직접 조작할 수 있습니다.

---

4. 특정 영역에서만 요소를 찾고 싶을 때 – 부분 범위 지정

전체 `<li>` 태그를 다 가져오는 것이 아니라, 특정 `ul` 안의 `<li>`만 가져오고 싶다면 반드시 **상위 요소를 기준으로 다시 호출**해야 합니다.

HTML 구조 예시:

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

<ul id="footer">
  <li>Contact</li>
  <li>Privacy</li>
</ul>
```

→ `#menu` 안의 `<li>`만 선택하려면 다음과 같이 작성합니다:

```
const menu = document.getElementById("menu");
const menuItems = menu.getElementsByTagName("li");
```

→ 이제 `menuItems`에는 `#menu` 내부의 두 개의 `<li>`만 포함되며, `#footer`의 항목은 포함되지 않습니다.

→ 이렇게 **상위 요소를 기준으로 한정하여 선택**하면 코드의 안정성과 효율성이 높아집니다.

---

5. 전체 태그 요소를 다 선택하고 싶을 때 – `"*"` 활용

```
const all = document.getElementsByTagName("*");
console.log(all.length); // 문서 전체 태그 수 출력
```

→ `*`를 사용하면 문서 내의 **모든 태그 요소**가 선택됩니다.

→ DOM 구조를 분석하거나 자동화 스크립트에서 페이지 전체를 탐색할 때 유용하지만, 성능 부담이 있으므로 꼭 필요한 경우에만 사용하세요.

---

6. 실시간 반영의 특징 – 요소 추가 후 길이 변화

```
const list = document.getElementsByTagName("li");

const newItem = document.createElement("li");
newItem.textContent = "새 항목 추가!";
document.querySelector("ul").appendChild(newItem);

console.log(list.length); // 자동으로 갱신됨
```

→ 처음 `list`를 가져올 때는 기존 항목만 있었지만, 새로운 `<li>`를 추가하자마자 `list.length`가 즉시 증가합니다.

→ 이것이 `HTMLCollection`이 일반 배열과 다르게 **DOM 상태를 실시간 반영**하는 특징입니다.

---

7. 일반 배열과 비교 – 수동 갱신과 자동 갱신의 차이

```
const arr = ["a", "b", "c"];
arr.splice(1, 1); // 'b' 제거 → ['a', 'c']
```

→ 일반 배열은 `splice()`와 같은 메서드로 직접 조작해야 하지만,

→ `HTMLCollection`은 요소를 DOM에서 추가/삭제하는 것만으로도 자동 갱신됩니다.

---

8. 실시간 갱신이 불러오는 문제 – 반복문 삭제 버그

```
const list = document.getElementsByTagName("li");
for (let i = 0; i < list.length; i++) {
  list[i].remove(); // ❌ 삭제 중 인덱스 밀림 발생
}
```

→ 삭제가 반복되는 동안 리스트가 계속 변경되기 때문에, 일부 요소가 **건너뛰어지고 삭제되지 않는 버그**가 생길 수 있습니다.

---

9. 해결 방법 – 정적 배열로 복사 후 안전하게 작업

```
const list = document.getElementsByTagName("li");
const copy = Array.from(list);

copy.forEach((item) => item.remove()); // ✅ 모든 항목 안정적으로 삭제
```

→ `Array.from()`으로 복사하면 live 속성을 제거한 정적인 배열이 되기 때문에, 삭제 중에도 인덱스 문제가 발생하지 않습니다.

✅ 핵심 요약

- `getElementsByTagName()`은 **특정 태그명을 기준으로 모든 요소를 선택**하는 메서드입니다.
- 반환값은 배열처럼 보이지만, 실제로는 **live한 HTMLCollection**입니다.
- 이 구조는 DOM에 변화가 생기면 자동으로 최신 상태로 반영되지만, 반복문 삭제 시 예기치 않은 동작을 유발할 수 있습니다.
- 안전한 조작이 필요할 경우에는 `Array.from()`으로 복사하여 정적인 배열로 처리하세요.
