15. 📘 HTMLCollection vs NodeList – 닮은 듯 다른 두 리스트의 정체

1. 겉보기는 똑같은데 동작은 전혀 다른 두 친구

우리는 `getElementsByTagName()`이나 `querySelectorAll()`처럼 여러 개의 요소를 한꺼번에 선택하는 메서드를 자주 사용합니다. 버튼 여러 개를 선택해서 색깔을 바꾸고 싶을 때나, 여러 개의 카드나 리스트 항목에 이벤트를 걸고 싶을 때 자주 등장하는 도구입니다. 그런데 이들이 반환하는 값은 이름부터 조금 생소한 `HTMLCollection` 또는 `NodeList`라는 객체입니다.

→ 둘 다 브라우저 콘솔에서 보면 **배열처럼 생겼고**, `[0]`, `[1]`처럼 인덱스로 개별 접근도 가능하며, `.length` 속성도 가지고 있습니다.

→ 하지만 실제로 웹페이지를 만들고 DOM을 조작하다 보면, 두 객체는 동작 방식이 아주 다르다는 것을 알게 됩니다. DOM이 바뀌는 상황에서, 어떤 것은 자동으로 최신 상태를 반영하는데, 어떤 것은 그대로 멈춰 있기도 하죠.

→ 이 차이를 모르면, 반복문으로 요소를 삭제했을 때 의도치 않게 일부가 남거나, 새로 추가한 요소가 반영되지 않는 등의 **예상치 못한 버그**가 생길 수 있습니다.

---

2. 어떤 메서드를 쓰느냐에 따라 결과가 달라진다

아래 표는 대표적인 DOM 선택 메서드와 반환값 타입을 정리한 것입니다:

선택 메서드	반환 타입
`getElementsByTagName()`	`HTMLCollection`
`getElementsByClassName()`	`HTMLCollection`
`document.forms`, `document.images`	`HTMLCollection`
`querySelectorAll()`	`NodeList`
`childNodes`	`NodeList`

→ 즉, 우리가 사용하는 DOM API에 따라, 똑같이 요소 여러 개를 반환하더라도 **어떤 타입의 리스트로 반환되는지 달라집니다.**

→ 이게 중요한 이유는 바로 **DOM 변화에 대한 반응성**, 즉 실시간으로 업데이트되는가 여부 때문입니다.

---

3. 기본 예제로 차이를 직접 확인해보기

다음과 같은 간단한 HTML을 가정해봅시다:

```
<div id="container">
  <p>첫 번째</p>
  <p>두 번째</p>
</div>
```

이제 자바스크립트로 위의 `<p>` 태그들을 각각 `HTMLCollection`과 `NodeList`로 가져와보겠습니다.

```
const tags1 = document.getElementsByTagName("p"); // HTMLCollection
const tags2 = document.querySelectorAll("p");     // NodeList
```

→ 둘 다 `<p>` 태그 2개를 선택하지만, 실제로 반환된 객체의 타입은 서로 다릅니다.

```
console.log(tags1); // HTMLCollection(2) [p, p]
console.log(tags2); // NodeList(2) [p, p]
```

→ 겉보기에 둘 다 유사 배열(배열처럼 생겼지만 배열은 아닌 객체)이고, 인덱스로 접근하거나 `.length`로 개수도 확인 가능하지만, **이 둘의 진짜 차이는 DOM 변경 이후에 드러납니다.**

---

4. DOM이 바뀌면? 실시간 반영 여부가 갈린다

예를 들어, `<p>` 태그를 하나 더 만들어서 `#container` 안에 동적으로 추가해보겠습니다:

```
const newP = document.createElement("p");
newP.textContent = "세 번째";
document.getElementById("container").appendChild(newP);

console.log(tags1.length); // 3
console.log(tags2.length); // 2
```

→ `tags1` (HTMLCollection)은 DOM이 변경되자마자 길이가 3으로 바뀌었습니다. 실시간으로 변화를 감지하고 반영하는 구조입니다.

→ 반면 `tags2` (NodeList)는 처음 선택했던 상태를 고정된 스냅샷처럼 유지하기 때문에 여전히 2입니다.

📌 **결론**:

- `HTMLCollection`은 실시간 반영되는 **라이브 컬렉션**
- `NodeList`는 선택 시점의 상태만 유지하는 **정적 컬렉션**

---

5. 삭제 시도에서 인덱스 밀림 문제가 생긴다

아래는 `<li>` 요소를 가져와서 삭제하려는 예시입니다:

```
<ul>
  <li>철수</li>
  <li>지민</li>
  <li>영희</li>
</ul>
```

```
const list = document.getElementsByTagName("li");
for (let i = 0; i < list.length; i++) {
  list[i].remove();
}
```

→ 예상과는 달리 `"지민"`이 삭제되지 않고 남는 문제가 발생합니다.

왜냐하면 `list`는 HTMLCollection이기 때문에, 요소를 하나 지우면 **자동으로 그 뒤 요소가 앞으로 당겨지기 때문**입니다.

예를 들어:

- i = 0 → `철수` 삭제 → `"지민"`이 [0]번 인덱스로 당겨짐
- i = 1 → `list[1]`은 `"영희"` → `"지민"`은 삭제 못하고 건너뜀

---

6. 해결 방법 1 – 배열로 복사해서 안정적으로 사용하기

가장 안정적인 방법은 아예 **배열로 복사해서** 사용하는 것입니다:

```
const items = Array.from(document.getElementsByTagName("li"));
items.forEach((li) => {
  li.remove();
});
```

→ 복사된 배열은 DOM과 연결되어 있지 않기 때문에, 요소를 삭제하더라도 **인덱스가 흔들리지 않습니다.**

→ `forEach`로 반복도 안정적으로 가능하고, `map`, `filter` 같은 배열 전용 메서드도 활용할 수 있습니다.

---

7. 해결 방법 2 – 뒤에서부터 반복하기

또 다른 방법은 **거꾸로 반복**하는 것입니다:

```
const list = document.getElementsByTagName("li");
for (let i = list.length - 1; i >= 0; i--) {
  list[i].remove();
}
```

→ 뒤에서부터 삭제하면, 앞쪽 인덱스들이 당겨져도 이미 지난 인덱스이므로 영향을 받지 않습니다.

→ 아주 단순하면서도 실무에서 자주 쓰이는 안전한 패턴입니다.

---

8. NodeList는 반복 중에도 안정적이다

반면 `querySelectorAll()`로 가져온 `NodeList`는 삭제 중에도 흔들리지 않습니다:

```
const list = document.querySelectorAll("li");
list.forEach((li) => li.remove());
```

→ `NodeList`는 선택 당시의 상태만 기억하므로, 그걸 기준으로 반복해도 삭제가 모두 잘 일어납니다.

→ 반복 중 DOM이 바뀌어도, 선택된 대상이 고정되어 있기 때문에 안전하게 사용 가능합니다.

9. 하지만 NodeList도 갱신되지 않는 단점이 있다

아래 예제를 보겠습니다:

```
const items = document.querySelectorAll(".item");

items.forEach((item) => {
  const newItem = document.createElement("div");
  newItem.className = "item";
  document.body.appendChild(newItem);
});

console.log(items.length); // 변화 없음
```

→ 여기서는 `.item` 클래스를 가진 요소들을 반복하면서 새 `.item`을 계속 추가하고 있지만,

→ `items`라는 NodeList는 처음 선택된 시점의 상태만 유지하기 때문에, **아무리 새로 추가해도 그 내용은 반영되지 않습니다.**

✅ 핵심 요약

- `HTMLCollection`은 DOM이 바뀌면 **자동으로 최신 상태를 반영**하는 라이브 컬렉션입니다.
- `NodeList`는 선택 당시의 상태를 유지하는 **정적 컬렉션**입니다.
- 반복문 중 요소를 삭제할 때는 **HTMLCollection은 인덱스 밀림 문제**가 발생할 수 있으므로 주의해야 합니다.
- 해결법으로는 `Array.from()`으로 복사하거나, **거꾸로 반복**, 또는 **NodeList 사용**이 있습니다.
- 반면 `NodeList`는 반복 중 새 요소 추가 시 **자동 반영되지 않기 때문에** 실시간 DOM 감지에는 부적합합니다.

→ 결국 어떤 상황이냐에 따라 더 적합한 선택지가 달라집니다. **정적인 조작은 NodeList, 동적인 조작은 HTMLCollection**이 어울릴 수 있습니다.
