14. 📘 querySelectorAll – 여러 요소를 유연하게 선택하는 도구

1. 반복되는 구조 속에서 필요한 도구

이제 우리는 단일 요소를 선택하는 다양한 방법들을 익혔고, 지난 강의에서는 CSS 선택자 문법을 그대로 사용할 수 있는 `querySelector()`를 배웠습니다.

하지만 실제 웹페이지는 이렇게 단순하지 않습니다.

→ 여러 개의 버튼, 카드, 이미지, 목록 등 **같은 클래스명과 구조를 가진 요소들이 반복해서 등장**합니다.

→ 이런 상황에서 단일 요소만 선택하는 방식은 너무 불편하고 비효율적입니다.

→ 바로 이럴 때 사용하는 것이 `querySelectorAll()`입니다.

---

2. 함수 이름 그대로 직역하면 이해된다

`querySelectorAll()`은 이름을 그대로 해석하면 다음과 같습니다:

- **query** → 찾다
- **selector** → 선택자
- **All** → 모두

→ 즉, "선택자에 해당하는 모든 요소를 찾아라!" 라는 뜻이 됩니다.

→ 이전에 배운 `querySelector()`가 **첫 번째 요소 하나만** 반환했다면,

→ `querySelectorAll()`은 **조건에 맞는 모든 요소를 한꺼번에 반환**합니다.

---

3. 기본 사용법 – 클래스명으로 여러 요소 선택하기

예를 들어 `.box` 클래스를 가진 모든 요소를 선택하고 싶다면 이렇게 작성합니다:

```
const boxes = document.querySelectorAll(".box");
```

→ HTML 구조는 다음과 같을 수 있습니다:

```
<div class="box">첫 번째 박스</div>
<div class="box">두 번째 박스</div>
<div class="box">세 번째 박스</div>
```

→ 이렇게 하면 세 개의 `.box` 요소들이 전부 선택되어 `boxes`라는 변수에 담깁니다.

---

4. 반환값은 NodeList – 배열처럼 보이지만 배열은 아니다

`querySelectorAll()`의 반환값은 **NodeList**라는 유사 배열입니다.

- 번호(index)로 접근 가능: `boxes[0]`, `boxes[1]` 등
- `forEach()`는 사용 가능
- 하지만 `map()`, `filter()` 같은 **진짜 배열 메서드는 사용 불가**

→ 마치 음식 메뉴판은 있지만 바로 주문은 안 되는 셀프바 메뉴판처럼, **제한된 기능을 가진 배열**이라고 생각하면 됩니다.

---

5. 반복 작업에 최적화 – forEach로 스타일 일괄 적용

선택된 요소들에 같은 작업을 하고 싶을 때 가장 많이 쓰이는 방식은 `forEach()`입니다:

```
boxes.forEach((box) => {
  box.style.border = "2px solid red";
});
```

→ 위 코드는 `.box` 클래스를 가진 모든 요소에 빨간 테두리를 적용합니다.

→ CSS에서 `.box { border: 2px solid red; }` 라고 한 것과 비슷한 효과를 JS로 주는 방식입니다.

---

6. 조합 선택자도 그대로 사용 가능하다

CSS에서 쓰던 선택자 문법을 그대로 활용할 수 있다는 점은 여전히 유효합니다:

```
const highlightedItems = document.querySelectorAll("#list .highlight");
```

→ 위 코드는 `id="list"`인 요소 내부에 있는 `.highlight` 클래스를 가진 모든 요소를 선택합니다.

→ HTML 구조 예시:

```
<div id="list">
  <p class="highlight">강조 1</p>
  <p class="highlight">강조 2</p>
  <p>일반 문장</p>
</div>
```

→ 결과적으로 두 개의 `<p class="highlight">` 요소가 선택됩니다.

---

7. 콘솔에서 구조 확인 – NodeList가 어떻게 생겼는지 보기

직접 확인하고 싶다면 다음과 같이 `console.dir()`을 사용하세요:

```
console.dir(document.querySelectorAll(".item"));
```

→ NodeList로 반환된 요소들의 태그 구조, 속성 목록 등을 브라우저 콘솔에서 한눈에 볼 수 있습니다.

---

8. 중요한 주의사항 – 동적으로 바뀌는 요소는 자동 반영되지 않는다

`querySelectorAll()`은 조건에 맞는 **모든 요소를 즉시 한 번만 선택**합니다.

→ 그 후 HTML 구조가 바뀌어도 자동 갱신되지 않습니다.

→ 반면 `getElementsByClassName()`은 **HTML이 변경되면 자동으로 최신 상태를 반영**합니다.

```
const dynamicItems = document.getElementsByClassName("dynamic");
// 이건 실시간 반영됨
```

→ 상황에 따라 어떤 선택자를 쓸지 판단해야 합니다.

---

9. 속성 선택자도 지원 – 조건이 있는 태그들만 골라내기

아래처럼 **속성 값을 조건으로 걸어주는 선택자**도 사용할 수 있습니다:

```
const inputs = document.querySelectorAll("input[type='text']");
```

→ HTML 예시:

```
<input type="text" />
<input type="password" />
<input type="text" />
```

→ 이 코드는 `type="text"`인 `<input>` 요소들만 NodeList로 선택합니다.

→ CSS에서 `[type="text"]`라고 쓰던 문법 그대로입니다.

✅ 핵심 요약

- `querySelectorAll()`은 CSS 선택자 문법을 그대로 사용하여 **모든 일치 요소들을 NodeList로 반환**합니다.
- 반환된 NodeList는 `forEach()`를 사용할 수 있지만 **배열은 아닙니다.**
- HTML이 바뀌더라도 자동 반영되지 않기 때문에, 동적 조작에는 주의가 필요합니다.
- 속성 선택자, 조합 선택자 등 CSS 문법을 그대로 사용할 수 있어 매우 직관적입니다.

→ 반복되는 DOM 조작이 필요한 순간, 가장 강력하고 직관적인 선택 메서드입니다.
