18. 📘 querySelector와 querySelectorAll, 언제 어떻게 쓸까

1. `querySelector()` 와 `querySelectorAll()` – 직관적인 CSS 선택자의 세계

우리는 앞서 `getElementById`, `getElementsByClassName`, `getElementsByTagName` 같은 메서드들을 배워왔습니다.

하지만 이 방식들은 단순 조건에서는 강력해도, **조금만 복잡한 구조나 조건이 생기면 유연성이 떨어집니다**.

바로 이 문제를 해결해주는 것이 `querySelector()`와 `querySelectorAll()`입니다.

→ 이 두 메서드는 **CSS 선택자 문법**을 그대로 사용할 수 있습니다.

→ 클래스, 태그, 아이디, 자손, 속성 등 모든 CSS 조합을 활용할 수 있어 매우 정밀한 탐색이 가능합니다.

---

2. 기본 구조 – 클래스와 자손 요소 선택

다음 HTML 구조를 보겠습니다:

```
<div class="card">
  <h2 class="title">첫 번째 카드</h2>
  <p class="desc">이것은 첫 번째 카드입니다.</p>
</div>
<div class="card">
  <h2 class="title">두 번째 카드</h2>
  <p class="desc">이것은 두 번째 카드입니다.</p>
</div>
```

첫 번째 카드의 제목만 선택하고 싶다면?

```
const firstTitle = document.querySelector(".card .title");
console.log(firstTitle.textContent); // 첫 번째 카드
```

→ `.card .title`은 `.card` 요소 내부에 있는 `.title`을 의미합니다.

→ 자손 선택자이며, 가장 먼저 나오는 첫 번째 매칭 요소 **하나만** 반환합니다.

---

3. 여러 요소를 선택할 때 – `querySelectorAll()` 과 `NodeList`

```
const allTitles = document.querySelectorAll(".card .title");

allTitles.forEach((el) => {
  console.log(el.textContent);
});
// 출력:
// 첫 번째 카드
// 두 번째 카드
```

→ `.card .title`에 해당하는 모든 요소를 찾아 `NodeList`로 반환합니다.

→ `NodeList`는 유사 배열이며, `forEach()`를 사용할 수 있어 배열처럼 다루기 편리합니다.

---

4. 쇼핑몰 예시 – 제품명 가져오기

```
<ul>
  <li class="product"><span class="name">에어팟</span></li>
  <li class="product"><span class="name">갤럭시버즈</span></li>
  <li class="product"><span class="name">헤드폰</span></li>
</ul>
```

```
const products = document.querySelectorAll(".product .name");
products.forEach((p) => console.log(p.textContent));
```

→ `.product` 클래스 내부의 `.name` 요소만 가져오는 방식. 실무에서도 자주 사용됩니다.

---

5. 구조 선택자 – 순서를 기준으로 선택하기

HTML에서 같은 구조가 반복될 때, 위치에 따라 특정 요소만 선택하려면 **구조 선택자**를 사용합니다.

```
const firstCardTitle = document.querySelector(".card:nth-of-type(1) .title");
console.log(firstCardTitle.textContent); // 첫 번째 카드
```

→ `:nth-of-type(n)`은 같은 태그 중 **n번째 요소**를 선택할 때 사용됩니다.

```
const secondCard = document.querySelector(".card:nth-of-type(2)");
```

→ 뉴스 목록에서 특정 기사에만 광고 배너를 삽입할 때 이런 방식이 유용합니다.

---

6. 속성 선택자 – 조건을 더 정밀하게

다음 구조를 보겠습니다.

```
<ul>
  <li class="item" data-id="1">Item 1</li>
  <li class="item" data-id="2">Item 2</li>
  <li class="item" data-id="3" hidden>Item 3</li>
</ul>
```

```
const secondItem = document.querySelector('.item[data-id="2"]');
console.log(secondItem.textContent); // Item 2
```

→ `[속성명="값"]` 형식의 선택자는 매우 강력합니다.

```
const saleItems = document.querySelectorAll('.product[data-category="sale"]');
saleItems.forEach((el) => el.classList.add("highlight"));
```

→ `data-*` 속성과 함께 쓰면, 필터링·분류·조건부 스타일링이 훨씬 유연해집니다.

---

7. 특정 요소 기준의 상대 탐색

```
const card = document.querySelector(".card"); // 첫 번째 카드
const desc = card.querySelector(".desc");
```

```
console.log(desc.textContent); // 이것은 첫 번째 카드입니다.
```

→ `document` 전체가 아닌 **특정 요소 기준**으로도 탐색할 수 있습니다.

→ 이렇게 하면 복잡한 페이지에서도 **컴포넌트 단위로 탐색을 제한**할 수 있어 효율적입니다.

---

8. 정적인 NodeList vs 실시간 HTMLCollection

`querySelectorAll()`의 반환값은 **정적인 NodeList**입니다.

```
const list = document.querySelectorAll(".item");
const newLi = document.createElement("li");
newLi.className = "item";
newLi.textContent = "Item 4";
document.querySelector("ul").appendChild(newLi);

console.log(list.length); // 여전히 3
```

→ DOM이 바뀌어도 NodeList는 자동으로 갱신되지 않습니다.

→ 따라서 변경 후엔 `querySelectorAll()`을 다시 호출해야 합니다.

```
button.addEventListener("click", () => {
  const items = document.querySelectorAll(".item"); // 새로 탐색
  items.forEach((el) => el.classList.add("updated"));
});
```

→ 반면 `getElementsByClassName()`은 **live collection**이므로 자동 반영됨.

---

9. 언제 어떤 선택자를 써야 하나?

목적	추천 메서드	특징
특정 ID 요소 하나 선택	`getElementById()`	가장 빠르고 명확
여러 클래스 요소 선택	`getElementsByClassName()`	live collection, 빠름
태그 기반 전체 선택	`getElementsByTagName()`	live collection

목적	추천 메서드	특징
조건이 복합적일 때	`querySelector()` / `querySelectorAll()`	CSS 문법 사용 가능, 정적 리스트

✅ 핵심 요약

- `querySelector()`는 CSS 문법으로 **첫 번째 매칭 요소** 하나를 반환.
- `querySelectorAll()`은 **모든 매칭 요소**를 `NodeList` 형태로 반환.
- `NodeList`는 정적이며 DOM이 바뀌어도 자동 갱신되지 않음.
- 선택자는 자손(`.a .b`), 속성(`[data-id="1"]`), 구조(`:nth-of-type(n)`) 등 자유롭게 조합 가능.
- **특정 요소 기준**으로도 탐색 가능하며, 컴포넌트 구조에서 매우 유용.
