

# 24. 📖 클래스와 스타일을 다루는 가장 강력한 방법 – classList, style, 그리고 리플로우까지

## 1. 눈에 보이는 화면을 바꾸는 진짜 힘

이제 우리는 DOM 요소의 속성을 어떻게 조회하고 조작하는지를 완전히 이해했습니다. 이번 시간부터는 그 중에서도 특히 **스타일링과 관련된 속성**, 즉 요소의 `class` 속성과 `style` 속성을 자바스크립트로 어떻게 다루는지를 심층적으로 살펴보겠습니다.

화면을 구성하는 HTML과 CSS는 **정적인 구조와 디자인**을 책임지고, 이 화면을 **동적으로 바꾸는 일은 자바스크립트의 역할**입니다. 예를 들어 버튼을 눌렀을 때 어떤 요소가 숨겨지거나, 특정 영역이 강조되거나, 테마가 전환되는 모든 동작은 **클래스 또는 스타일 속성 조작**을 통해 이루어 집니다.

## 2. 클래스 조작의 시작: `className` vs `classList`

HTML에서는 다음처럼 클래스를 정의합니다:

```
<div class="box active"></div>
```

이 요소를 JS로 제어하려면?

### 🔪 방법 1: `className`

```
const box = document.querySelector(".box");
console.log(box.className); // "box active"
box.className = "box highlight"; // 전체 교체
```

→ **단점**: 문자열 전체를 수정해야 하므로 `"active"` 만 제거하거나 `"highlight"` 만 추가하는 것이 번거롭고 위험합니다.

## 3. 해결책 등장! `classList` 의 정체

`classList` 는 클래스 이름들을 마치 배열처럼 다룰 수 있게 해주는 **DOMTokenList 객체**입니다. 내부적으로는 `"box active highlight"` 를 `"box"` , `"active"` , `"highlight"` 처럼 **토큰(token)** 으로 분리해서 다룹니다.

## 4. classList 주요 메서드 정리

```
const box = document.querySelector(".box");
```

### ✓ add()

```
box.classList.add("active");
```

→ 해당 클래스를 추가. 이미 있으면 중복 없이 유지.

### ✓ remove()

```
box.classList.remove("hidden");
```

→ 해당 클래스를 제거. 없으면 아무 일도 없음.

### ✓ toggle()

```
box.classList.toggle("highlight");
```

→ 이미 있으면 제거, 없으면 추가. 스위치처럼 동작.

### ✓ contains()

```
box.classList.contains("active"); // true / false
```

→ 해당 클래스가 존재하는지 확인 (조건 분기용).

## 5. 실전 예시: 메뉴 열기/닫기

```
const menuBtn = document.querySelector("#menuBtn");
const menu = document.querySelector("#sideMenu");

menuBtn.addEventListener("click", () => {
  menu.classList.toggle("open");
});
```

→ 클릭할 때마다 `.open` 클래스가 토글되며 메뉴가 열리고 닫힘.

## 6. 스타일 조작의 핵심: `style` 객체

### 직접 인라인 스타일 설정

```
const box = document.querySelector(".box");
box.style.backgroundColor = "red";
box.style.border = "1px solid black";
```

→ HTML에 다음처럼 반영됨:

```
<div class="box" style="background-color: red; border: 1px solid black;"></div>
```

## 7. 자주 쓰는 스타일 속성명 (JS용 표기법)

CSS 명령어	JavaScript 표기
background-color	backgroundColor
font-size	fontSize
z-index	zIndex

✅ 하이픈(-) 대신 카멜표기법으로 사용!

## 8. 성능 고려: 클래스 vs 인라인 스타일

❌ 성능 나쁜 방식 (리플로우 유발)

```
items.forEach(item => {
  item.style.backgroundColor = "yellow";
  item.style.fontWeight = "bold";
});
```

→ 직접 인라인 스타일을 루프마다 지정 = 리페인트/리플로우 증가



## ✓ 성능 좋은 방식

```
/* style.css */
.highlight {
  background-color: yellow;
  font-weight: bold;
}
```

```
items.forEach(item => {
  item.classList.add("highlight");
});
```

→ 클래스 변경으로 렌더링 성능 향상 + 코드 재사용성 증가

## 9. 렌더링 성능 이슈 – 리페인트 vs 리플로우

구분	설명
 리페인트	색상, 배경, 테두리 등 <b>겉모습만</b> 다시 그림
 리플로우	크기, 위치, 레이아웃 등 <b>배치 전체를 재계산</b>

→ 리플로우는 더 비용이 큼니다. 따라서 스타일은 되도록 클래스 조작으로 처리하는 것이 이상적입니다.

## ✓ 핵심 요약

- `className` 은 문자열 전체를 다루므로 조심해서 사용해야 합니다.
- `classList` 는 배열처럼 클래스를 추가/제거/토글할 수 있는 안전한 방식입니다.

- `style` 객체를 통해 인라인 스타일을 직접 적용할 수 있지만, 성능상 권장되지 않습니다.
  - 반복 적용 시엔 CSS 클래스를 정의하고 `classList` 로 관리하는 것이 훨씬 효율적입니다.
  - 브라우저 성능에 영향을 주는 **리플로우** 개념을 이해하고 최적화에 활용해야 합니다.
-