Config는 왜 필요한가: 데이터와 렌더링 사이의 간극

1. 문제 상황

API에서 데이터가 온다고 가정하자.

```
{ load: 85, battery: 72, name: "UPS-001" }
```

이 데이터를 화면에 표시하려면 무엇이 필요할까?

숫자 85는 그 자체로는 아무 의미가 없다. 코드는 다음을 알아야 한다:

- 이 값을 어느 DOM 요소에 넣을 것인가?
- 단위가 있는가? (%, W, ℃)
- 변환이 필요한가? (소수점, 천단위 콤마)

데이터는 "값"만 제공한다. "값을 어떻게 표시할지"는 제공하지 않는다.

---

2. Config의 역할

Config는 이 간극을 메운다.

```
const config = {
    fields: [
        { key: 'load', selector: '.ups-load', suffix: '%' },
        { key: 'battery', selector: '.ups-battery', suffix: '%' },
        { key: 'name', selector: '.ups-name' }
    ]
};
```

이제 렌더링 함수는 다음을 수행할 수 있다:

1. `key: 'load'` → 데이터에서 `data.load` 값(85)을 꺼낸다
2. `selector: '.ups-load'` → DOM에서 해당 요소를 찾는다
3. `suffix: '%'` → "85%"로 표시한다

---

3. Config 없이 코드를 작성하면

```
function renderUPS(data) {
    document.querySelector('.ups-load').textContent = data.load + '%';
    document.querySelector('.ups-battery').textContent = data.battery + '%';
    document.querySelector('.ups-name').textContent = data.name;
}
```

문제점:

상황	결과
필드가 추가되면	함수 수정
selector가 바뀌면	함수 수정
단위가 바뀌면	함수 수정

다른 화면에서 재사용	새 함수 작성

렌더링 로직과 화면 구조가 **강하게 결합**되어 있다.

---

4. Config가 있으면

```
function renderFields(config, data) {
    config.fields.forEach(field => {
        const el = document.querySelector(field.selector);
        if (!el) return;
        let value = data[field.key] ?? '-';
        if (field.suffix) value += field.suffix;
        el.textContent = value;
    });
}
```

변경 시:

상황	수정 대상
필드 추가	config만 수정
selector 변경	config만 수정
단위 변경	config만 수정
다른 화면 재사용	새 config 작성

렌더링 함수는 건드리지 않는다.

---

5. Config의 정체

Config에 담기는 정보의 공통점:

- **HTML 구조** → 개발자가 작성했으므로 안다
- **API 필드명** → API 명세에서 안다
- **표시 형식** → 기획/디자인에서 안다

모두 **코드 작성 시점에 알 수 있는 정보**다.

반면 `85`, `72` 같은 실제 값은 **런타임에만 알 수 있다**.

구분	예시	알 수 있는 시점
Config	selector, key, suffix	개발 시점
Data	85, 72, "UPS-001"	런타임

Config는 "사전 지식"이고, Data는 "실시간 정보"다.

---

6. 요약

데이터만으로는 렌더링할 수 없다. **데이터를 해석하는 방법**이 필요하다.

구분	역할	질문

Data	무엇을 (What)	값
Config	어디에, 어떻게 (Where, How)	해석 방법

Config는 개발 시점에 알 수 있는 정보를 선언적으로 분리한 것이다.

이 분리가 코드 재사용성과 유지보수성을 만든다.

관련 문서

- README.md - 부록 D: Configuration 설계 원칙