컴포넌트 이벤트 처리 원칙

1. 핵심 판단 기준

"이 동작의 결과를 페이지가 알아야 하는가?"

이 질문이 이벤트 처리 방식을 결정한다.

페이지가 알아야 하는가?	처리 방식	예시
아니오	`_internalHandlers` + `addEventListener`	토글, 확장/축소, 내부 UI 상태
예	`customEvents` + `bindEvents()`	행 선택, 필터 변경, 버튼 클릭
둘 다	둘 다 사용	클릭 → UI 변경 + 페이지 알림

---

2. customEvents + bindEvents

2.1 용도

페이지가 알아야 하는 이벤트를 선언적으로 정의하고 Weventbus로 발행한다.

2.2 선언 형식

```
this.customEvents = {
    'click': {
        '.btn-apply': '@filterApplied',
        '.btn-reset': '@filterReset'
    }
};

bindEvents(this, this.customEvents);
```

2.3 내부 동작 (Wkit.js:240-254)

```
function makeHandler(targetInstance, selector) {
  return function (event) {
    const { customEvents } = targetInstance;
    const triggerEvent = customEvents?.[event.type]?.[selector];
    if (triggerEvent) {
      Weventbus.emit(triggerEvent, {
        event,           // DOM 이벤트 객체
        targetInstance,  // 컴포넌트 인스턴스 전체
      });
    }
  };
}
```

2.4 페이지에서 수신

```
this.eventBusHandlers = {
    '@filterApplied': ({ event, targetInstance }) => {
        // event: DOM 이벤트 객체
```

```
        // targetInstance: 컴포넌트 인스턴스 전체

        // 컴포넌트 내부 상태 접근 가능
        const filters = targetInstance._currentFilters;

        // param 업데이트 및 재조회
        this.currentParams['tasks'] = { ...filters };
        GlobalDataPublisher.fetchAndPublish('tasks', this, this.currentParams['tasks']);
    }
};
```

2.5 전달되는 것

속성	내용	접근 예시
event	DOM 이벤트 객체	event.target, event.type
targetInstance	컴포넌트 인스턴스 전체	targetInstance._currentFilters, targetInstance.name

중요: `targetInstance`로 컴포넌트 인스턴스 전체가 전달되므로, 페이지에서 컴포넌트 내부 상태(`_currentFilters` 등)에 접근 가능하다.

---

3. _internalHandlers

3.1 용도

페이지가 알 필요 없는 내부 UI 동작을 처리한다.

3.2 패턴

```
function setupInternalHandlers() {
    const root = this.appendElement;
    const ctx = this;

    this._internalHandlers = {
        // 내부 상태만 업데이트 (페이지 알림 불필요)
        selectChange: (e) => {
            const filterType = e.target.dataset.filter;
            if (filterType) {
                ctx._currentFilters[filterType] = e.target.value;
            }
        },

        // UI 초기화 (페이지 알림은 customEvents가 담당)
        resetClick: () => {
            ctx._currentFilters = { status: 'all', priority: 'all' };
            root.querySelectorAll('.filter-select').forEach(select => {
                select.value = 'all';
            });
        }
    };

    // 직접 addEventListener로 등록
    root.querySelectorAll('.filter-select').forEach(select => {
        select.addEventListener('change', this._internalHandlers.selectChange);
```

```
    });
}
```

3.3 적합한 상황

- 셀렉트 변경 시 내부 상태 업데이트
- 토글 버튼으로 UI 확장/축소
- 입력 필드 값 임시 저장
- 드롭다운 열기/닫기

---

4. 둘 다 사용하는 케이스

하나의 이벤트가 **내부 UI 동작**과 **페이지 알림** 둘 다 필요한 경우.

4.1 예시: Reset 버튼

```
// customEvents: 페이지에 알림
this.customEvents = {
    'click': {
        '.btn-reset': '@filterReset'
    }
};

// _internalHandlers: 내부 UI 초기화
this._internalHandlers = {
    resetClick: () => {
        // 내부 상태 초기화
        ctx._currentFilters = { status: 'all', priority: 'all' };

        // 셀렉트 UI 초기화
        root.querySelectorAll('.filter-select').forEach(select => {
            select.value = 'all';
        });
    }
};

// 같은 요소에 두 핸들러 모두 등록
root.querySelector('.btn-reset')?.addEventListener('click', this._internalHandlers.resetClick);
```

동작 순서:

1. 클릭 발생
2. `_internalHandlers.resetClick` 실행 → UI 초기화
3. `customEvents` 의 delegate 핸들러 실행 → `@filterReset` 발행
4. 페이지의 `eventBusHandlers['@filterReset']` 실행

---

5. 생성/정리 매칭

생성 (register)	정리 (beforeDestroy)
`this.customEvents = {...}`	`this.customEvents = null`
`bindEvents(this, customEvents)`	`removeCustomEvents(this, customEvents)`
`this._internalHandlers = {...}`	`this._internalHandlers = null`

el.addEventListener(type, handler)	el.removeEventListener(type, handler)

5.1 beforeDestroy 예시

```
const { removeCustomEvents } = Wkit;

// 1. customEvents 해제
removeCustomEvents(this, this.customEvents);
this.customEvents = null;

// 2. 내부 이벤트 해제
if (this._internalHandlers) {
    const root = this.appendElement;

    root.querySelectorAll('.filter-select').forEach(select => {
        select.removeEventListener('change', this._internalHandlers.selectChange);
    });

    root.querySelector('.btn-reset')?.removeEventListener('click', this._internalHandlers.resetClick);

    this._internalHandlers = null;
}
```

---

6. 판단 체크리스트

새로운 이벤트를 추가할 때:

- ☐ 이 동작의 결과를 페이지가 알아야 하는가?
- ☐ 알아야 한다면 → `customEvents`
- ☐ 알 필요 없다면 → `_internalHandlers`
- ☐ 둘 다라면 → 둘 다 사용
- ☐ beforeDestroy에서 정리 코드가 매칭되어 있는가?

---

7. 흔한 실수

7.1 customEvents로 내부 상태 변경만 하는 경우

```
// ❌ 잘못된 패턴: 페이지가 알 필요 없는데 customEvents 사용
this.customEvents = {
    'change': {
        '.filter-select': '@selectChanged'  // 페이지에서 이 이벤트를 쓰지 않음
    }
};

// 페이지
'@selectChanged': ({ targetInstance }) => {
    // 아무것도 안 함 - 의미 없는 이벤트 발행
}
```

```
// ✅ 올바른 패턴: 내부 상태만 변경하므로 _internalHandlers 사용
this._internalHandlers = {
```

```
    selectChange: (e) => {
        this._currentFilters[e.target.dataset.filter] = e.target.value;
    }
};
```

7.2 중복 바인딩

```
// ❌ 잘못된 패턴: 같은 동작을 두 번 등록
this.customEvents = {
    'click': { '.btn-apply': '@filterApplied' }
};
bindEvents(this, this.customEvents);

// 또 등록
root.querySelector('.btn-apply').addEventListener('click', () => {
    Weventbus.emit('@filterApplied', { ... });  // 중복!
});
```

---

8. 요약

1. **판단 기준**: "페이지가 알아야 하는가?"
2. **예** → `customEvents` + `bindEvents()`
3. **아니오** → `_internalHandlers` + `addEventListener`
4. **둘 다** → 둘 다 사용
5. **정리**: 생성한 것은 반드시 매칭하여 해제

관련 문서

- README.md - 부록 C: 컴포넌트 내부 이벤트 패턴
- README.md - 컴포넌트 라이프사이클 패턴