이벤트 처리

Vue School의 무료 동영상 강의 보기

이벤트 리스닝

우리는 `v-on` 디렉티브(일반적으로 `@` 기호로 축약)를 사용하여 DOM 이벤트를 리스닝하고, 이벤트가 발생할 때 JavaScript를 실행할 수 있습니다. 사용법은 `v-on:click="handler"` 또는 축약형인 `@click="handler"` 입니다.

핸들러 값은 다음 중 하나일 수 있습니다:

1. **인라인 핸들러:** 이벤트가 발생할 때 실행될 인라인 JavaScript(네이티브 `onclick` 속성과 유사).
2. **메서드 핸들러:** 컴포넌트에 정의된 메서드를 가리키는 속성 이름 또는 경로.

인라인 핸들러

인라인 핸들러는 일반적으로 간단한 경우에 사용됩니다. 예를 들어:

```js
const count = ref(0)
```

```template
<button @click="count++">Add 1</button>
<p>Count is: {{ count }}</p>
```

Playground에서 실행해보기

메서드 핸들러

많은 이벤트 핸들러의 로직은 더 복잡할 수 있으며, 인라인 핸들러로는 적합하지 않을 수 있습니다. 그래서 `v-on` 은 호출하고자 하는 컴포넌트 메서드의 이름이나 경로도 받을 수 있습니다.

예를 들어:

```js
const name = ref('Vue.js')

function greet(event) {
  alert(`Hello ${name.value}!`)
  // `event`는 네이티브 DOM 이벤트입니다
  if (event) {
    alert(event.target.tagName)
  }
}
```

```template
<!-- `greet`는 위에 정의된 메서드의 이름입니다 -->
<button @click="greet">Greet</button>
```

Playground에서 실행해보기

메서드 핸들러는 자동으로 해당 이벤트를 발생시킨 네이티브 DOM Event 객체를 전달받습니다. 위 예제에서는 `event.target` 을 통해 이벤트를 발생시킨 엘리먼트에 접근할 수 있습니다.

참고: 이벤트 핸들러 타입 지정 TS

메서드 vs. 인라인 감지

템플릿 컴파일러는 `v-on` 값 문자열이 유효한 JavaScript 식별자 또는 속성 접근 경로인지 확인하여 메서드 핸들러를 감지합니다. 예를 들어, `foo` , `foo.bar` , `foo['bar']` 는 메서드 핸들러로 처리되고, `foo()` 와 `count++` 는 인라인 핸들러로 처리됩니다.

인라인 핸들러에서 메서드 호출하기

메서드 이름에 직접 바인딩하는 대신, 인라인 핸들러에서 메서드를 호출할 수도 있습니다. 이를 통해 네이티브 이벤트 대신 메서드에 커스텀 인자를 전달할 수 있습니다:

```js
function say(message) {
  alert(message)
```

```template
<button @click="say('hello')">Say hello</button>
<button @click="say('bye')">Say bye</button>
```

Playground에서 실행해보기

인라인 핸들러에서 이벤트 인자 접근하기

때때로 인라인 핸들러에서 원래의 DOM 이벤트에 접근해야 할 때가 있습니다. 이럴 때는 특별한 `$event` 변수를 사용하거나, 인라인 화살표 함수를 사용할 수 있습니다:

```template
<!-- $event 특수 변수를 사용 -->
<button @click="warn('Form cannot be submitted yet.', $event)">
  Submit
</button>

<!-- 인라인 화살표 함수 사용 -->
<button @click="(event) => warn('Form cannot be submitted yet.', event)">
  Submit
</button>
```

```js
function warn(message, event) {
  // 이제 네이티브 이벤트에 접근할 수 있습니다
  if (event) {
    event.preventDefault()
  }
  alert(message)
}
```

이벤트 수식자

이벤트 핸들러 내부에서 `event.preventDefault()` 나 `event.stopPropagation()` 을 호출해야 하는 경우가 매우 흔합니다. 물론 메서드 내부에서 쉽게 할 수 있지만, 메서드가 DOM 이벤트 세부 사항을 처리하지 않고 데이터 로직에만 집중할 수 있다면 더 좋을 것입니다.

이 문제를 해결하기 위해 Vue는 `v-on` 에 **이벤트 수식자**를 제공합니다. 수식자는 점으로 표시되는 디렉티브 접미사입니다.

```
.stop
.prevent
```

`.capture`

`.once`

`.passive`

template

```
<!-- 클릭 이벤트의 전파가 중단됩니다 -->
<a @click.stop="doThis"></a>

<!-- submit 이벤트가 더 이상 페이지를 새로고침하지 않습니다 -->
<form @submit.prevent="onSubmit"></form>

<!-- 수식자는 체이닝할 수 있습니다 -->
<a @click.stop.prevent="doThat"></a>

<!-- 수식자만 사용 -->
<form @submit.prevent></form>

<!-- event.target이 해당 엘리먼트일 때만 핸들러가 실행됩니다 -->
<!-- 즉, 자식 엘리먼트에서 발생한 이벤트는 제외 -->
<div @click.self="doThat">...</div>
```

> ⓘ TIP
>
> 수식자를 사용할 때는 순서가 중요합니다. 관련 코드는 같은 순서로 생성되기 때문입니다. 따라서 `@click.prevent.self` 는 **해당 엘리먼트와 자식 모두에서 클릭의 기본 동작을 막고**, `@click.self.prevent` 는 해당 엘리먼트에서만 클릭의 기본 동작을 막습니다.

`.capture` , `.once` , `.passive` 수식자는 네이티브 `addEventListener` 메서드의 옵션과 동일하게 동작합니다:

template

```
<!-- 이벤트 리스너를 추가할 때 캡처 모드 사용     -->
<!-- 즉, 내부 엘리먼트를 타겟팅한 이벤트가       -->
<!-- 해당 엘리먼트에서 먼저 처리됩니다           -->
<div @click.capture="doThis">...</div>

<!-- 클릭 이벤트는 최대 한 번만 트리거됩니다 -->
<a @click.once="doThis"></a>

<!-- 스크롤 이벤트의 기본 동작(스크롤링)은 `onScroll`이 완료될 때까지 기다리지 않고    -->
<!-- 즉시 발생합니다. `event.preventDefault()`가  -->
<!-- 포함되어 있을 경우에도 마찬가지입니다.      -->
<div @scroll.passive="onScroll">...</div>
```

`.passive` 수식자는 일반적으로 모바일 기기에서 성능 향상을 위해 터치 이벤트 리스너와 함께 사용됩니다.

TIP

`.passive` 와 `.prevent` 를 함께 사용하지 마세요. `.passive` 는 이미 브라우저에 이벤트의 기본 동작을 막지 않을 것임을 알리므로, 함께 사용하면 브라우저에서 경고가 발생할 수 있습니다.

키 수식자

키보드 이벤트를 리스닝할 때, 특정 키를 확인해야 하는 경우가 많습니다. Vue는 키 이벤트를 리스닝할 때 `v-on` 또는 `@` 에 키 수식자를 추가할 수 있습니다:

template

```
<!-- `key`가 `Enter`일 때만 `submit` 호출 -->
<input @keyup.enter="submit" />
```

`KeyboardEvent.key` 에서 노출되는 유효한 키 이름을 케밥 케이스로 변환하여 수식자로 직접 사용할 수 있습니다.

template

```
<input @keyup.page-down="onPageDown" />
```

위 예제에서 핸들러는 `$event.key` 가 `'PageDown'` 일 때만 호출됩니다.

키 별칭

Vue는 자주 사용되는 키에 대해 별칭을 제공합니다:

- `.enter`
- `.tab`
- `.delete` ("Delete"와 "Backspace" 키 모두 캡처)
- `.esc`
- `.space`
- `.up`
- `.down`
- `.left`
- `.right`

시스템 수정 키

거되도록 할 수 있습니다:

- `.ctrl`
- `.alt`
- `.shift`
- `.meta`

> ⓘ 참고
>
> Macintosh 키보드에서 meta는 command 키(⌘)입니다. Windows 키보드에서는 meta가 Windows 키(⊞)입니다. Sun Microsystems 키보드에서는 meta가 실선 다이아몬드(◆)로 표시됩니다. MIT 및 Lisp 머신 키보드와 그 후속 키보드(예: Knight 키보드, space-cadet 키보드)에서는 meta가 “META”로 표시됩니다. Symbolics 키보드에서는 meta가 “META” 또는 “Meta”로 표시됩니다.

예를 들어:

template

```
<!-- Alt + Enter -->
<input @keyup.alt.enter="clear" />

<!-- Ctrl + Click -->
<div @click.ctrl="doSomething">Do something</div>
```

> ⓘ TIP
>
> 수정 키는 일반 키와 다르며, `keyup` 이벤트와 함께 사용할 때는 이벤트가 발생할 때 반드시 눌려 있어야 합니다. 즉, `keyup.ctrl` 은 ctrl을 누른 상태에서 다른 키를 뗄 때만 트리거됩니다. ctrl 키만 뗄 때는 트리거되지 않습니다.

`.exact` 수식자

`.exact` 수식자를 사용하면 이벤트를 트리거하는 데 필요한 시스템 수정 키의 정확한 조합을 제어할 수 있습니다.

template

```
<!-- Alt 또는 Shift가 함께 눌려도 실행됩니다 -->
<button @click.ctrl="onClick">A</button>

<!-- Ctrl만 눌렸을 때만 실행됩니다 -->
<button @click.ctrl.exact="onCtrlClick">A</button>

<!-- 시스템 수정 키가 아무것도 눌리지 않았을 때만 실행됩니다 -->
<button @click.exact="onClick">A</button>
```

마우스 버튼 수식자

- `.left`
- `.right`
- `.middle`

이 수식자들은 특정 마우스 버튼으로 트리거된 이벤트에만 핸들러를 제한합니다.

단, `.left`, `.right`, `.middle` 수식자 이름은 일반적인 오른손잡이 마우스 레이아웃을 기준으로 하지만, 실제로는 각각 "주", "보조", "보조2" 포인팅 장치 이벤트 트리거를 의미하며, 실제 물리적 버튼과는 다를 수 있습니다. 예를 들어, 왼손잡이 마우스 레이아웃에서는 "주" 버튼이 실제로 오른쪽 버튼일 수 있지만 `.left` 수식자 핸들러가 트리거됩니다. 트랙패드의 경우 한 손가락 탭은 `.left`, 두 손가락 탭은 `.right`, 세 손가락 탭은 `.middle` 핸들러를 트리거할 수 있습니다. 이와 같이, "마우스" 이벤트를 생성하는 다른 장치나 이벤트 소스도 "left"와 "right"와는 무관한 트리거 모드를 가질 수 있습니다.

GitHub에서 이 페이지 편집

Previous
리스트 렌더링

Next
폼 입력 바인딩