이벤트 핸들링

이벤트 리스닝하기

일반적으로 v-on 디렉티브는 단축 문법으로 @ 기호를 사용하며, DOM 이벤트를 수신하고 트리거될 때, 사전 정의해둔 JavaScript 코드를 실행할 수 있습니다. v-on:click="handler" 또는 줄여서 @click="handler" 와 같이 사용합니다.

핸들러 값은 다음 중 하나일 수 있습니다:

인라인 핸들러: 이벤트가 트리거될 때 실행되는 인라인 JavaScript(네이티브 onclick 속성과 유사).

메서드 핸들러: 컴포넌트에 정의된 메서드 이름 또는 메서드를 가리키는 경로.

인라인 핸들러

인라인 핸들러는 일반적으로 다음과 같이 간단한 경우에 사용됩니다:

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

```template
<button @click="count++">1 추가</button>
<p>숫자 값은: {{ count }}</p>
```

온라인 연습장으로 실행하기

메서드 핸들러

대부분의 이벤트 핸들러 논리는 복잡할 것이며, 인라인 핸들러에서는 실현 가능하지 않을 수 있습니다. 그러므로 v-on 은 컴포넌트의 메서드 이름이나 메서드를 가리키는 경로를 실행할 수 있게 구현되어 있습니다.

예제:

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

function greet(event) {
  alert(`안녕 ${name.value}!`)
  // 'event'는 네이티브 DOM 이벤트 객체입니다.
  if (event) {
    alert(event.target.tagName)
  }
}
```

```template
<!-- `greet`는 위에서 정의한 메서드의 이름입니다. -->
<button @click="greet">환영하기</button>
```

온라인 연습장으로 실행하기

메서드 핸들러는 이를 트리거하는 네이티브 DOM 이벤트 객체를 자동으로 수신합니다. 위의 예에서 event.target 을 통해 이벤트를 전달하는 엘리먼트에 접근할 수 있습니다.

참고: 이벤트 핸들러 타입 지정하기

메서드 vs 인라인 구분

템플릿 컴파일러는 `v-on` 값인 문자열이 유효한 JavaScript 식별자 또는 속성에 접근 가능한 경로인지를 확인해 메서드 핸들러를 감지합니다. 예를 들어 `foo`, `foo.bar` 및 `foo['bar']` 는 메서드 핸들러로 처리되는 반면, `foo()` 및 `count++` 는 인라인 핸들러로 처리됩니다.

인라인 핸들러에서 메서드 호출하기

메서드 이름을 직접 바인딩하는 대신, 인라인 핸들러에서 메서드를 호출할 수도 있습니다. 이를 통해 네이티브 이벤트 객체 대신 사용자 지정 인자를 메서드에 전달할 수 있습니다:

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

```template
<button @click="say('안녕')">안녕이라고 말하기</button>
<button @click="say('잘가')">잘가라고 말하기</button>
```

온라인 연습장으로 실행하기

인라인 핸들러에서 이벤트 객체 접근하기

때로는 인라인 핸들러에서 네이티브 DOM 이벤트 객체에 접근해야 하는 경우도 있습니다. 특수한 키워드인 `$event` 를 사용하여 메서드에 전달하거나 인라인 화살표 함수를 사용할 수 있습니다:

```template
<!-- 특수한 키워드인 $event 사용 -->
<button @click="warn('아직 양식을 제출할 수 없습니다.', $event)">
  제출하기
</button>

<!-- 인라인 화살표 함수 사용 -->
<button @click="(event) => warn('아직 양식을 제출할 수 없습니다.', event)">
  제출하기
</button>
```

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

이벤트 수식어

이벤트 핸들러 내에서 `event.preventDefault()` 또는 `event.stopPropagation()` 을 호출하는 것은 매우 흔한 일입니다. 메서드 내에서 이 작업을 쉽게 수행할 수 있지만, 메서드가 DOM 이벤트에 대한 세부적인 처리 로직 없이 오로지 데이터 처리 로직만 있다면 코드 유지보수

에 더 좋을 것입니다.

이 문제를 해결하기 위해 v-on 은 점(.)으로 시작하는 지시적 접미사인 **이벤트 수식어**를 제공합니다.

.stop
.prevent
.self
.capture
.once
.passive

template

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

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

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

<!-- 이벤트에 핸들러 없이 수식어만 사용할 수 있습니다. -->
<form @submit.prevent></form>

<!-- event.target이 엘리먼트 자신일 경우에만 핸들러가 실행됩니다. -->
<!-- 예를 들어 자식 엘리먼트에서 클릭 액션이 있으면 핸들러가 실행되지 않습니다. -->
<div @click.self="doThat">...</div>
```

TIP

수정자를 사용할 때는 관련 코드가 동일한 순서로 생성되므로 순서가 중요합니다. 따라서 @click.prevent.self 를 사용하면 **엘리먼트 자체와 그 자식에 대한 클릭의 기본 동작**을 방지하는 반면, @click.self.prevent 는 엘리먼트 자체에 대한 클릭의 기본 동작만 방지합니다.

.capture , .once 및 .passive 수식어는 네이티브 **addEventListener** 메서드의 옵션을 반영합니다:

template

```
<!-- 이벤트 리스너를 추가할 때 캡처 모드 사용 -->
<!-- 내부 엘리먼트에서 클릭 이벤트 핸들러가 실행되기 전에, -->
<!-- 여기에서 먼저 핸들러가 실행됩니다. -->
<div @click.capture="doThis">...</div>

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

<!-- 핸들러 내 `event.preventDefault()`가 포함되었더라도 -->
<!-- 스크롤 이벤트의 기본 동작(스크롤)이 발생합니다. -->
<div @scroll.passive="onScroll">...</div>
```

.passive 수식어는 일반적으로 모바일 장치의 성능 향상을 위해 터치 이벤트 리스너와 함께 사용됩니다.

TIP

.passive 는 브라우저에게 이벤트의 _기본 동작을 방지(prevent)하지 않겠다_는 의도를 전달한 것입니다. 따라서 .passive 와 .prevent 를 함께 사용해서는 안되며, 그렇게 할 경우 브라우저(콘솔)에서 경고 메세지를 볼 수 있습니다.

입력키 수식어

키보드 이벤트를 수신할 때, 특정 키를 확인해야 하는 경우가 많기 때문에, 키 수식어를 지원합니다:

template

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

KeyboardEvent.key 를 통해 유효한 입력키 이름을 kebab-case로 변환하여 수식어로 사용할 수 있습니다:

template

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

위의 예에서 핸들러는 `$event.key` 가 `'PageDown'` 일 경우에만 호출됩니다:

입력키 별칭

Vue는 가장 일반적으로 사용되는 키에 대한 별칭을 제공합니다:

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

시스템 입력키 수식어

마우스 또는 키보드 이벤트 리스너는 아래 수식어를 사용하여, 해당 입력키를 누를 때만 트리거 되도록 할 수 있습니다:

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

> 참고
>
> Macintosh 키보드에서 메타는 커멘드 키(⌘)입니다. Windows 키보드에서 meta는 Windows 키(⊞)입니다. Sun Microsystems 키보드에서 메타는 실선 다이아몬드(◆)로 표시됩니다. Knight 키보드, space-cadet 키보드와 같은 특정 키보드, 특히 MIT 및 Lisp 기계 키보드 및 후속 제품에서는 메타가 "META"로 표시됩니다. Symbolics 키보드에서 메타는 "META" 또는 "Meta"로 표시됩니다.

예:

template

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

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

> TIP
>
> 시스템 수식어 입력키는 일반 키와 다르므로 `keyup` 이벤트와 함께 사용되는 경우, 키가 눌려져 있어야 이벤트가 발생합니다. 즉, `keyup.ctrl` 은 `ctrl` 을 누른 상태에서 다른 입력키를 땔 때(key up)만 트리거됩니다. `ctrl` 키만 땔 때는 트리거되지 않습니다.

.exact 수식어

.exact 수식어를 사용하면 이벤트를 트리거하는 데 필요한 시스템 수식어의 정확한 조합을 제어할 수 있습니다.

```
<!-- Ctrl과 함께 Alt 또는 Shift를 누른 상태에서도 클릭하면 실행됩니다. -->
<button @click.ctrl="onClick">A</button>

<!-- 오직 Ctrl만 누른 상태에서 클릭해야 실행됩니다. -->
<button @click.ctrl.exact="onCtrlClick">A</button>

<!-- 시스템 입력키를 누르지 않고 클릭해야지만 실행됩니다. -->
<button @click.exact="onClick">A</button>
```

마우스 버튼 수식어

- .left
- .right
- .middle

특정 마우스 버튼에 의해 이벤트가 트리거 되도록 제한하고 싶을 때 사용합니다.