컴포넌트 이벤트

> 이 페이지는 이미 컴포넌트 기본을 읽었다고 가정합니다. 컴포넌트가 처음이라면 먼저 해당 내용을 읽으세요.

이벤트 발생 및 리스닝

컴포넌트는 내장된 `$emit` 메서드를 사용하여 템플릿 표현식(예: `v-on` 핸들러)에서 직접 커스텀 이벤트를 발생시킬 수 있습니다:

template

```
<!-- MyComponent -->
<button @click="$emit('someEvent')">Click Me</button>
```

부모는 `v-on` 을 사용하여 해당 이벤트를 리스닝할 수 있습니다:

template

```
<MyComponent @some-event="callback" />
```

컴포넌트 이벤트 리스너에서도 `.once` 수식어를 사용할 수 있습니다:

template

```
<MyComponent @some-event.once="callback" />
```

컴포넌트와 props처럼, 이벤트 이름도 자동으로 케이스 변환이 적용됩니다. 우리는 camelCase 이벤트를 발생시켰지만, 부모에서는 kebab-case 리스너로 리스닝할 수 있습니다. props 케이스와 마찬가지로, 템플릿에서는 kebab-case 이벤트 리스너 사용을 권장합니다.

TIP

식 컴포넌트가 발생시킨 이벤트만 리스닝할 수 있습니다. 형제 또는 깊게 중첩된 컴포넌트 간에 통신이 필요하다면 외부 이벤트 버스나 글로벌 상태 관리 솔루션을 사용하세요.

이벤트 인자

이벤트와 함께 특정 값을 발생시키는 것이 유용할 때가 있습니다. 예를 들어, `<BlogPost>` 컴포넌트가 텍스트를 얼마나 확대할지 결정하도록 하고 싶을 수 있습니다. 이런 경우, `$emit` 에 추가 인자를 전달하여 값을 제공할 수 있습니다:

template

```
<button @click="$emit('increaseBy', 1)">
  1만큼 증가
</button>
```

그런 다음, 부모에서 이벤트를 리스닝할 때 인라인 화살표 함수를 리스너로 사용하면 이벤트 인자에 접근할 수 있습니다:

template

```
<MyButton @increase-by="(n) => count += n" />
```

또는, 이벤트 핸들러가 메서드인 경우:

template

```
<MyButton @increase-by="increaseCount" />
```

그러면 그 값이 해당 메서드의 첫 번째 매개변수로 전달됩니다:

js

```
function increaseCount(n) {
  count.value += n
}
```

> ⓘ TIP
>
> 이벤트 이름 뒤에 `$emit()` 에 전달된 모든 추가 인자는 리스너로 전달됩니다. 예를 들어, `$emit('foo', 1, 2, 3)` 의 경우 리스너 함수는 세 개의 인자를 받게 됩니다.

발생시킬 이벤트 선언하기

컴포넌트는 `defineEmits()` 매크로을 사용하여 발생시킬 이벤트를 명시적으로 선언할 수 있습니다:

```vue
<script setup>
defineEmits(['inFocus', 'submit'])
</script>
```

`<template>` 에서 사용한 `$emit` 메서드는 컴포넌트의 `<script setup>` 섹션 내에서는 접근할 수 없지만, `defineEmits()` 는 대신 사용할 수 있는 동등한 함수를 반환합니다:

```vue
<script setup>
const emit = defineEmits(['inFocus', 'submit'])

function buttonClick() {
  emit('submit')
}
</script>
```

`defineEmits()` 매크로는 **함수 내부에서 사용할 수 없으며**, 위 예시처럼 반드시 `<script setup>` 내에 직접 위치해야 합니다.

명시적인 `setup` 함수를 `<script setup>` 대신 사용하는 경우, 이벤트는 `emits` 옵션을 사용해 선언해야 하며, `emit` 함수는 `setup()` 컨텍스트에서 노출됩니다:

```js
export default {
  emits: ['inFocus', 'submit'],
  setup(props, ctx) {
    ctx.emit('submit')
  }
}
```

`setup()` 컨텍스트의 다른 속성과 마찬가지로, `emit` 도 안전하게 구조 분해 할당할 수 있습니다:

```js
export default {
  emits: ['inFocus', 'submit'],
  setup(props, { emit }) {
    emit('submit')
  }
}
```

우 인자에 타입을 지정할 수 있어, 발생시킨 이벤트의 페이로드에 대한 런타임 유효성 검사가 가능합니다:

```vue
<script setup lang="ts">
const emit = defineEmits({
  submit(payload: { email: string, password: string }) {
    // 유효성 검사 통과/실패를 나타내기 위해
    // `true` 또는 `false`를 반환합니다.
  }
})
</script>
```

`<script setup>` 에서 TypeScript를 사용하는 경우, 순수 타입 주석만으로도 발생시킬 이벤트를 선언할 수 있습니다:

```vue
<script setup lang="ts">
const emit = defineEmits<{
  (e: 'change', id: number): void
  (e: 'update', value: string): void
}>()
</script>
```

자세한 내용: 컴포넌트 Emits 타입 지정 TS

선택 사항이지만, 컴포넌트가 어떻게 동작해야 하는지 더 잘 문서화하기 위해 모든 발생시킬 이벤트를 정의하는 것이 좋습니다. 또한 Vue가 전달 속성에서 알려진 리스너를 제외할 수 있게 하여, 서드파티 코드가 수동으로 디스패치한 DOM 이벤트로 인한 예외적인 상황을 방지할 수 있습니다.

> ⓘ TIP
>
> 네이티브 이벤트(예: `click`)가 `emits` 옵션에 정의되어 있으면, 리스너는 이제 컴포넌트에서 발생시킨 `click` 이벤트만 리스닝하며, 더 이상 네이티브 `click` 이벤트에는 반응하지 않습니다.

이벤트 유효성 검사

props 타입 유효성 검사와 유사하게, 발생시킬 이벤트가 배열 문법이 아닌 객체 문법으로 정의된 경우 유효성 검사를 할 수 있습니다.

부를 boolean으로 반환하는 함수를 할당합니다.

vue

```vue
<script setup>
const emit = defineEmits({
  // 유효성 검사 없음
  click: null,

  // submit 이벤트 유효성 검사
  submit: ({ email, password }) => {
    if (email && password) {
      return true
    } else {
      console.warn('유효하지 않은 submit 이벤트 페이로드입니다!')
      return false
    }
  }
})

function submitForm(email, password) {
  emit('submit', { email, password })
}
</script>
```

GitHub에서 이 페이지 편집

Previous
Props

Next
컴포넌트 v-model