컴포넌트 이벤트

이 페이지는 이미 컴포넌트 기초를 읽었다고 가정합니다. 컴포넌트에 익숙하지 않다면 먼저 그것을 읽어보세요.

이벤트 발생 및 수신

컴포넌트는 내장 `$emit` 메소드를 사용하여 템플릿 표현식에서 직접 사용자 정의 이벤트를 발생시킬 수 있습니다(예: `v-on` 핸들러에서):

template

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

부모는 `v-on` 을 사용하여 이를 수신할 수 있습니다:

template

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

컴포넌트 이벤트 리스너에서 `.once` 수정자도 지원됩니다:

template

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

컴포넌트와 props와 마찬가지로, 이벤트 이름은 자동으로 대소문자 변환을 제공합니다. camelCase 이벤트를 발생시켰지만, 부모에서 kebab-case 리스너를 사용하여 수신할 수 있습니다. **props casing**와 마찬가지로, 템플릿에서 kebab-case 이벤트 리스너를 사용하는 것이 좋습니다.

> **TIP**
>
> 네이티브 **DOM** 이벤트와 달리, 컴포넌트에서 발생한 이벤트는 **버블링되지 않습니다**. 직접 자식 컴포넌트에서 발생한 이벤트만 수신할 수 있습니다. 형제 컴포넌트나 깊숙이 중첩된 컴포넌트 간에 통신이 필요한 경우, 외부 이벤트 버스 또는 글로벌 상태 관리 솔루션을 사용하세요.

이벤트 인수

이벤트와 함께 특정 값을 발생시키는 것이 유용할 때가 있습니다. 예를 들어, `<BlogPost>` 컴포넌트가 텍스트를 얼마나 확대할지 결정하게 하고 싶을 수 있습니다. 이 경우, 이 값을 제공하기 위해 `$emit` 에 추가 인수를 전달할 수 있습니다:

template

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

그런 다음 부모에서 이벤트를 수신할 때, 인라인 화살표 함수를 리스너로 사용할 수 있으며, 이를 통해 이벤트 인수에 접근할 수 있습니다:

template

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

또는 이벤트 핸들러가 메소드인 경우:

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

그러면 해당 메소드의 첫 번째 매개변수로 값이 전달됩니다:

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

> **TIP**
>
> 이벤트 이름 다음에 `$emit()` 에 전달된 모든 추가 인수는 리스너에게 전달됩니다. 예를 들어, `$emit('foo', 1, 2, 3)` 으로 `$emit` 을 사용하면 리스너 함수는 세 개의 인수를 받게 됩니다.

발생하는 이벤트 선언하기

컴포넌트는 **defineEmits()** 매크로을 사용하여 발생할 이벤트를 명시적으로 선언할 수 있습니다:

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

`<template>` 에서 사용한 `$emit` 메소드는 컴포넌트의 `<script setup>` 섹션 내에서 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 내부가 아니라 `<script setup>` 에서 직접 사용해야 합니다.

명시적인 `setup` 함수를 사용하는 경우 이벤트는 **emits** 옵션을 사용하여 선언되어야 하며, `emit` 함수는 `setup()` 컨텍스트에 노출됩니다:

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

`setup()` 컨텍스트의 다른 속성과 마찬가지로, `emit` 은 안전하게 구조 분해할 수 있습니다:

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

```
  }
}
```

emits 옵션과 defineEmits() 매크로는 객체 구문도 지원합니다. TypeScript를 사용하는 경우 인수를 타입 지정할 수 있으며, 이를 통해 발생된 이벤트의 페이로드에 대한 런타임 유효성 검사를 수행할 수 있습니다:

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

<script setup> 을 사용하여 TypeScript와 함께 작업하는 경우, 순수한 타입 주석을 사용하여 발생된 이벤트를 선언할 수도 있습니다:

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

자세한 정보: 컴포넌트 이벤트 타이핑

이벤트를 정의하는 것은 선택 사항이지만, 컴포넌트가 어떻게 동작해야 하는지 더 잘 문서화하기 위해 모든 발생 이벤트를 정의하는 것이 좋습니다. 또한 이를 통해 Vue는 알려진 리스너를 낙하 속성에서 제외시켜 타사 코드에 의해 수동으로 발송된 DOM 이벤트에 의해 발생하는 에지 케이스를 피할 수 있습니다.

> **TIP**
>
> emits 옵션에서 네이티브 이벤트(예: click)를 정의한 경우, 리스너는 이제 네이티브 click 이벤트가 아니라 컴포넌트에서 발생한 click 이벤트에만 응답합니다.

이벤트 유효성 검사

prop 타입 유효성 검사와 마찬가지로, 배열 구문 대신 객체 구문으로 정의된 이벤트는 유효성 검사가 가능합니다.

유효성 검사를 추가하려면, emit 호출로 전달된 인수를 받는 함수에 이벤트를 할당하고, 이벤트가 유효한지 여부를 나타내는 부울을 반환합니다.

```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('Invalid submit event payload!')
      return false
    }
  }
})
```

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