Component v-model

기본 사용법

v-model 을 컴포넌트에서 사용하여 양방향 바인딩을 구현할 수 있습니다.

Vue 3.4부터는 **defineModel()** 매크로를 사용하는 것이 권장되는 접근 방식입니다:

vue

```
<!-- Child.vue -->
<script setup>
const model = defineModel()

function update() {
  model.value++
}
</script>

<template>
  <div>부모 바인딩 v-model은: {{ model }}</div>
</template>
```

부모는 v-model 을 사용하여 값을 바인딩할 수 있습니다:

template

```
<!-- Parent.vue -->
<Child v-model="countModel" />
```

defineModel() 에 의해 반환되는 값은 ref입니다. 다른 ref처럼 접근하고 변경할 수 있지만, 부모 값과 로컬 값 사이의 양방향 바인딩으로 작동합니다:

.value 는 부모 v-model 에 의해 바인딩된 값과 동기화됩니다;
자식에 의해 변경되면 부모 바인딩 값도 업데이트됩니다.

따라서 이 ref를 네이티브 입력 엘리먼트의 v-model 에 바인딩할 수도 있어, 네이티브 입력 엘리먼트를 래핑하면서 동일한 v-model 사용을 제공하는 것이 간단해집니다:

vue

```
<script setup>
const model = defineModel()
</script>

<template>
  <input v-model="model" />
</template>
```

Try it in the Playground

내부 구조

defineModel 은 편의성을 위한 매크로입니다. 컴파일러는 다음과 같이 확장합니다:

로컬 ref의 값과 동기화되는 modelValue 라는 이름의 prop;
로컬 ref의 값이 변경될 때 발생하는 update:modelValue 라는 이벤트.

3.4 이전에 위와 같은 자식 컴포넌트를 구현하는 방법은 다음과 같습니다:

```vue
<!-- Child.vue -->
<script setup>
const props = defineProps(['modelValue'])
const emit = defineEmits(['update:modelValue'])
</script>

<template>
  <input
    :value="props.modelValue"
    @input="emit('update:modelValue', $event.target.value)"
  />
</template>
```

그런 다음, 부모 컴포넌트에서 `v-model="modelValue"` 는 다음과 같이 컴파일됩니다:

```template
<!-- Parent.vue -->
<Child
  :modelValue="foo"
  @update:modelValue="$event => (foo = $event)"
/>
```

보시다시피, 이것은 훨씬 더 장황합니다. 하지만 내부에서 무슨 일이 일어나는지 이해하는 것이 도움이 됩니다.

`defineModel` 은 prop을 선언하므로, `defineModel` 에 전달함으로써 기본 prop의 옵션을 선언할 수 있습니다:

```js
// v-model을 필수로 만들기
const model = defineModel({ required: true })

// 기본값 제공
const model = defineModel({ default: 0 })
```

> **WARNING**
>
> `defineModel` `prop`에 `default` 값을 설정하고, 부모 컴포넌트에서 이 `prop`에 대한 값을 제공하지 않으면, 부모와 자식 컴포넌트 간의 동기화 문제가 발생할 수 있습니다. 아래 예시에서, 부모의 `myRef` 는 값이 정의되지 않았지만(`undefined`) 자식의 `model` 은 1 입니다:
>
> ```js
> // 자식 컴포넌트:
> const model = defineModel({ default: 1 })
>
> // 부모 컴포넌트:
> const myRef = ref()
> ```
>
> ```html
> <Child v-model="myRef"></Child>
> ```

v-model 인수

`v-model` 은 컴포넌트에서 인수를 받을 수도 있습니다:

```template
<MyComponent v-model:title="bookTitle" />
```

자식 컴포넌트에서는 defineModel() 의 첫 번째 인수로 문자열을 전달하여 해당 인수를 지원할 수 있습니다:

```vue
<!-- MyComponent.vue -->
<script setup>
const title = defineModel('title')
</script>

<template>
  <input type="text" v-model="title" />
</template>
```

Try it in the Playground

prop 옵션이 필요한 경우, 모델 이름 뒤에 전달해야 합니다:

```js
const title = defineModel('title', { required: true })
```

▼ 3.4 이전 사용법

```vue
<!-- MyComponent.vue -->
<script setup>
defineProps({
  title: {
    required: true
  }
})
defineEmits(['update:title'])
</script>

<template>
  <input
    type="text"
    :value="title"
    @input="$emit('update:title', $event.target.value)"
  />
</template>
```

Try it in the Playground

Multiple v-model bindings

앞서 배운 것처럼 특정 prop과 이벤트를 타깃팅하는 기능을 **v-model** 인자로 활용하면 이제 단일 컴포넌트 인스턴스에 여러 개의 v-model 바인딩을 생성할 수 있습니다.

각 v-model 은 컴포넌트에서 추가 옵션 없이도 다른 prop에 동기화됩니다:

```template
<UserName
  v-model:first-name="first"
  v-model:last-name="last"
/>
```

vue

```vue
<script setup>
const firstName = defineModel('firstName')
const lastName = defineModel('lastName')
</script>

<template>
  <input type="text" v-model="firstName" />
  <input type="text" v-model="lastName" />
</template>
```

Try it in the Playground

▼ 3.4 이전 사용법

vue

```vue
<script setup>
defineProps({
  firstName: String,
  lastName: String
})

defineEmits(['update:firstName', 'update:lastName'])
</script>

<template>
  <input
    type="text"
    :value="firstName"
    @input="$emit('update:firstName', $event.target.value)"
  />
  <input
    type="text"
    :value="lastName"
    @input="$emit('update:lastName', $event.target.value)"
  />
</template>
```

Try it in the Playground

v-model 수정자 처리하기

Form 양식 입력 바인딩에 대해 배울 때 v-model 에 .trim , .number 및 .lazy 와 같은 내장 수정자가 있다는 것을 알았습니다. 경우에 따라 사용자 정의 입력 컴포넌트에서 v-model 이 사용자 정의 수정자를 지원하도록 할 수도 있습니다.

v-model 바인딩에서 제공하는 문자열의 첫 글자를 대문자로 표시하는 사용자 지정 수정자 예제인 capitalize 를 만들어 보겠습니다:

template

```template
<MyComponent v-model.capitalize="myText" />
```

컴포넌트 v-model 에 추가된 수정자(modifiers)는 자식 컴포넌트에서 defineModel() 반환값을 구조 분해하여 다음과 같이 접근할 수 있습니다:

vue

```vue
<script setup>
const [model, modifiers] = defineModel()

console.log(modifiers) // { capitalize: true }
</script>
```

```vue
<template>
  <input type="text" v-model="model" />
</template>
```

수정자에 기반하여 값을 어떻게 읽거나 쓸지 조건적으로 조정하려면, `defineModel()` 에 `get` 과 `set` 옵션을 전달할 수 있습니다. 이 두 옵션은 모델 ref의 get / set에서 값을 받아 변환된 값을 반환해야 합니다. 다음은 `set` 옵션을 사용하여 `capitalize` 수정자를 구현하는 방법입니다:

```vue
<script setup>
const [model, modifiers] = defineModel({
  set(value) {
    if (modifiers.capitalize) {
      return value.charAt(0).toUpperCase() + value.slice(1)
    }
    return value
  }
})
</script>

<template>
  <input type="text" v-model="model" />
</template>
```

Try it in the Playground

▼ 3.4 이전 사용법

```vue
<script setup>
const props = defineProps({
  modelValue: String,
  modelModifiers: { default: () => ({}) }
})

const emit = defineEmits(['update:modelValue'])

function emitValue(e) {
  let value = e.target.value
  if (props.modelModifiers.capitalize) {
    value = value.charAt(0).toUpperCase() + value.slice(1)
  }
  emit('update:modelValue', value)
}
</script>

<template>
  <input type="text" :value="modelValue" @input="emitValue" />
</template>
```

Try it in the Playground

인자와 수정자가 있는 `v-model` 에 대한 수정자들

다음은 다른 인자를 가진 여러 `v-model` 에 수정자를 사용한 예시입니다:

```template
<UserName
  v-model:first-name.capitalize="first"
```

```
  v-model:last-name.uppercase="last"
/>
```

vue

```
<script setup>
const [firstName, firstNameModifiers] = defineModel('firstName')
const [lastName, lastNameModifiers] = defineModel('lastName')

console.log(firstNameModifiers) // { capitalize: true }
console.log(lastNameModifiers) // { uppercase: true }
</script>
```

▼ 3.4 이전 사용법

```
<script setup>
const props = defineProps({
  firstName: String,
  lastName: String,
  firstNameModifiers: { default: () => ({}) },
  lastNameModifiers: { default: () => ({}) }
})
defineEmits(['update:firstName', 'update:lastName'])

console.log(props.firstNameModifiers) // { capitalize: true }
console.log(props.lastNameModifiers) // { uppercase: true }
</script>
```