클래스와 스타일 바인딩

데이터 바인딩에서 흔히 필요한 것은 엘리먼트의 클래스 목록과 인라인 스타일을 조작하는 것입니다. `class` 와 `style` 은 모두 속성이기 때문에, 다른 속성들과 마찬가지로 `v-bind` 를 사용해 문자열 값을 동적으로 할당할 수 있습니다. 하지만 이러한 값을 문자열 결합으로 생성하려고 하면 번거롭고 오류가 발생하기 쉽습니다. 이런 이유로 Vue는 `v-bind` 를 `class` 와 `style` 에 사용할 때 특별한 기능을 제공합니다. 문자열뿐만 아니라, 해당 표현식이 객체나 배열로 평가될 수도 있습니다.

HTML 클래스 바인딩

Vue School의 무료 동영상 강의 보기

객체에 바인딩하기

`:class` (즉, `v-bind:class` 의 축약형)에 객체를 전달하여 클래스를 동적으로 토글할 수 있습니다:

```template
<div :class="{ active: isActive }"></div>
```

위의 문법은 `active` 클래스의 존재 여부가 데이터 속성 `isActive` 의 truthiness에 의해 결정됨을 의미합니다.

객체에 더 많은 필드를 추가하여 여러 클래스를 토글할 수 있습니다. 또한, `:class` 디렉티브는 일반 `class` 속성과 함께 사용할 수도 있습니다. 다음과 같은 상태가 있다고 가정해봅시다:

```js
const isActive = ref(true)
const hasError = ref(false)
```

```template
<div
  class="static"
  :class="{ active: isActive, 'text-danger': hasError }"
></div>
```

렌더링 결과는 다음과 같습니다:

```template
<div class="static active"></div>
```

`isActive` 나 `hasError` 가 변경되면 클래스 목록도 그에 맞게 업데이트됩니다. 예를 들어, `hasError` 가 `true` 가 되면 클래스 목록은 `"static active text-danger"` 가 됩니다.

바인딩하는 객체는 반드시 인라인일 필요는 없습니다:

```js
const classObject = reactive({
  active: true,
  'text-danger': false
})
```

```template
<div :class="classObject"></div>
```

이렇게 렌더링됩니다:

```template
<div class="active"></div>
```

계산된 속성에 객체를 반환하도록 바인딩할 수도 있습니다. 이는 일반적이고 강력한 패턴입니다:

```js
const isActive = ref(true)
const error = ref(null)

const classObject = computed(() => ({
  active: isActive.value && !error.value,
  'text-danger': error.value && error.value.type === 'fatal'
}))
```

```template
<div :class="classObject"></div>
```

배열에 바인딩하기

```js
const activeClass = ref('active')
const errorClass = ref('text-danger')
```

```template
<div :class="[activeClass, errorClass]"></div>
```

이렇게 렌더링됩니다:

```template
<div class="active text-danger"></div>
```

리스트 내의 클래스를 조건부로 토글하고 싶다면, 삼항 연산자를 사용할 수 있습니다:

```template
<div :class="[isActive ? activeClass : '', errorClass]"></div>
```

이렇게 하면 항상 `errorClass` 는 적용되고, `activeClass` 는 `isActive` 가 참일 때만 적용됩니다.

하지만 조건부 클래스가 여러 개라면 다소 장황해질 수 있습니다. 그래서 배열 문법 안에 객체 문법을 사용할 수도 있습니다:

```template
<div :class="[{ [activeClass]: isActive }, errorClass]"></div>
```

컴포넌트와 함께 사용하기

> 이 섹션은 컴포넌트에 대한 지식을 전제로 합니다. 건너뛰고 나중에 다시 와도 괜찮습니다.

컴포넌트에 `class` 속성을 사용하면, 해당 클래스들은 컴포넌트의 루트 엘리먼트에 추가되고 이미 존재하는 클래스와 병합됩니다.

예를 들어, `MyComponent` 라는 컴포넌트가 다음과 같은 템플릿을 가지고 있다면:

```template
<!-- 자식 컴포넌트 템플릿 -->
<p class="foo bar">Hi!</p>
```

사용 시에 클래스를 추가하면:

```template
<!-- 컴포넌트 사용 시 -->
<MyComponent class="baz boo" />
```

```template
<p class="foo bar baz boo">Hi!</p>
```

클래스 바인딩도 마찬가지입니다:

```template
<MyComponent :class="{ active: isActive }" />
```

`isActive` 가 참일 때, 렌더링된 HTML은 다음과 같습니다:

```template
<p class="foo bar active">Hi!</p>
```

컴포넌트에 루트 엘리먼트가 여러 개라면, 어떤 엘리먼트가 이 클래스를 받을지 정의해야 합니다. 이는 `$attrs` 컴포넌트 속성을 사용하여 할 수 있습니다:

```template
<!-- $attrs를 사용하는 MyComponent 템플릿 -->
<p :class="$attrs.class">Hi!</p>
<span>이것은 자식 컴포넌트입니다</span>
```

```template
<MyComponent class="baz" />
```

렌더링 결과:

```html
<p class="baz">Hi!</p>
<span>이것은 자식 컴포넌트입니다</span>
```

컴포넌트 속성 상속에 대한 더 자세한 내용은 속성 전달 섹션에서 확인할 수 있습니다.

인라인 스타일 바인딩

객체에 바인딩하기

`:style` 은 JavaScript 객체 값에 바인딩하는 것을 지원합니다. 이는 HTML 엘리먼트의 `style` 속성에 해당합니다:

```
const fontSize = ref(30)
```

template

```
<div :style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
```

카멜케이스 키를 권장하지만, `:style` 은 케밥 케이스 CSS 속성 키도 지원합니다(실제 CSS에서 사용하는 방식과 동일). 예를 들어:

template

```
<div :style="{ 'font-size': fontSize + 'px' }"></div>
```

템플릿을 더 깔끔하게 만들기 위해 스타일 객체에 직접 바인딩하는 것이 종종 좋습니다:

js

```
const styleObject = reactive({
  color: 'red',
  fontSize: '30px'
})
```

template

```
<div :style="styleObject"></div>
```

마찬가지로, 객체 스타일 바인딩은 객체를 반환하는 계산된 속성과 함께 자주 사용됩니다.

`:style` 디렉티브는 `:class` 와 마찬가지로 일반 style 속성과 함께 사용할 수 있습니다.

템플릿:

template

```
<h1 style="color: red" :style="'font-size: 1em'">hello</h1>
```

렌더링 결과:

template

```
<h1 style="color: red; font-size: 1em;">hello</h1>
```

배열에 바인딩하기

`:style` 에 여러 스타일 객체의 배열을 바인딩할 수 있습니다. 이 객체들은 병합되어 동일한 엘리먼트에 적용됩니다:

template

```
<div :style="[baseStyles, overridingStyles]"></div>
```

`:style` 에서 벤더 접두사가 필요한 CSS 속성을 사용할 때, Vue는 적절한 접두사를 자동으로 추가합니다. Vue는 런타임에 현재 브라우저에서 어떤 스타일 속성이 지원되는지 확인하여 이를 수행합니다. 브라우저가 특정 속성을 지원하지 않으면 다양한 접두사 버전을 테스트하여 지원되는 속성을 찾으려고 시도합니다.

여러 값

스타일 속성에 여러 개의 (접두사가 붙은) 값을 배열로 제공할 수 있습니다. 예를 들어:

```template
<div :style="{ display: ['-webkit-box', '-ms-flexbox', 'flex'] }"></div>
```

이 경우, 브라우저가 지원하는 배열의 마지막 값만 렌더링됩니다. 이 예시에서는 flexbox의 접두사 없는 버전을 지원하는 브라우저에서는 `display: flex` 가 렌더링됩니다.

GitHub에서 이 페이지 편집

Previous
계산된 속성

Next
조건부 렌더링