· 암스테르담 · Oct 09-10 등록하기

리스트 렌더링

Vue School의 무료 동영상 강의 보기

v-for

`v-for` 디렉티브를 사용하여 배열을 기반으로 항목 목록을 렌더링할 수 있습니다. `v-for` 디렉티브는 `item in items` 형태의 특별한 문법을 필요로 하며, 여기서 `items` 는 소스 데이터 배열이고 `item` 은 반복되는 배열 요소의 **별칭**입니다:

```js
const items = ref([{ message: 'Foo' }, { message: 'Bar' }])
```

```template
<li v-for="item in items">
  {{ item.message }}
</li>
```

`v-for` 의 스코프 내에서는 템플릿 표현식이 모든 부모 스코프 속성에 접근할 수 있습니다. 또한, `v-for` 는 현재 항목의 인덱스를 위한 두 번째 별칭도 지원합니다:

```js
const parentMessage = ref('Parent')
const items = ref([{ message: 'Foo' }, { message: 'Bar' }])
```

```template
<li v-for="(item, index) in items">
  {{ parentMessage }} - {{ index }} - {{ item.message }}
</li>
```

- Parent - 0 - Foo
- Parent - 1 - Bar

`v-for` 의 변수 스코프는 다음 JavaScript와 유사합니다:

```js
const parentMessage = 'Parent'
const items = [
  /* ... */
]

items.forEach((item, index) => {
  // 외부 스코프의 `parentMessage`에 접근 가능
  // 하지만 `item`과 `index`는 여기서만 사용 가능
  console.log(parentMessage, item.message, index)
})
```

`v-for` 값이 `forEach` 콜백의 함수 시그니처와 일치하는 것을 볼 수 있습니다. 실제로, 함수 인자 구조 분해와 유사하게 `v-for` 의 항목 별칭에서도 구조 분해를 사용할 수 있습니다:

```template
<li v-for="{ message } in items">
  {{ message }}
</li>

<!-- 인덱스 별칭과 함께 -->
<li v-for="({ message }, index) in items">
  {{ message }} {{ index }}
</li>
```

중첩된 `v-for` 의 경우, 스코프는 중첩 함수와 유사하게 동작합니다. 각 `v-for` 스코프는 부모 스코프에 접근할 수 있습니다:

```template
<li v-for="item in items">
  <span v-for="childItem in item.children">
    {{ item.message }} {{ childItem }}
  </span>
</li>
```

또한, 구분자로 `in` 대신 `of` 를 사용할 수도 있어, JavaScript의 반복자 문법과 더 유사하게 만들 수 있습니다:

```template
<div v-for="item of items"></div>
```

객체에 대한 `v-for`

`Object.values()` 를 호출한 결과를 기반으로 합니다:

```js
const myObject = reactive({
  title: 'How to do lists in Vue',
  author: 'Jane Doe',
  publishedAt: '2016-04-10'
})
```

```template
<ul>
  <li v-for="value in myObject">
    {{ value }}
  </li>
</ul>
```

속성의 이름(즉, 키)에 대한 두 번째 별칭도 제공할 수 있습니다:

```template
<li v-for="(value, key) in myObject">
  {{ key }}: {{ value }}
</li>
```

그리고 인덱스에 대한 또 다른 별칭도 사용할 수 있습니다:

```template
<li v-for="(value, key, index) in myObject">
  {{ index }}. {{ key }}: {{ value }}
</li>
```

플레이그라운드에서 직접 해보기

범위에 대한 `v-for`

`v-for` 는 정수도 받을 수 있습니다. 이 경우 `1...n` 범위에 따라 해당 횟수만큼 템플릿을 반복합니다.

```template
<span v-for="n in 10">{{ n }}</span>
```

여기서 `n` 은 0이 아닌 1부터 시작한다는 점에 유의하세요.

`<template>` 에서의 `v-for`

템플릿 `v-if` 와 유사하게, 여러 요소의 블록을 렌더링하기 위해 `<template>` 태그와 함께 `v-for` 를 사용할 수도 있습니다. 예를 들어:

template

```
<ul>
  <template v-for="item in items">
    <li>{{ item.msg }}</li>
    <li class="divider" role="presentation"></li>
  </template>
</ul>
```

`v-if` 와 함께 사용하는 `v-for`

동일한 노드에 존재할 때, `v-if` 가 `v-for` 보다 우선순위가 높습니다. 즉, `v-if` 조건은 `v-for` 의 스코프 변수에 접근할 수 없습니다:

template

```
<!--
이 코드는 "todo" 속성이 인스턴스에 정의되어 있지 않기 때문에
오류가 발생합니다.
-->
<li v-for="todo in todos" v-if="!todo.isComplete">
  {{ todo.name }}
</li>
```

이 문제는 `v-for` 를 감싸는 `<template>` 태그로 옮겨서 해결할 수 있습니다(이 방법이 더 명확하기도 합니다):

template

```
<template v-for="todo in todos">
  <li v-if="!todo.isComplete">
    {{ todo.name }}
  </li>
</template>
```

⚠ 참고

암묵적인 우선순위 때문에 `v-if` 와 `v-for` 를 동일한 요소에 사용하는 것은 **권장되지 않습니다**.

이렇게 하고 싶어지는 일반적인 두 가지 경우가 있습니다:

우, `users` 를 필터링된 리스트(예: `activeUsers`)를 만완아는 새로운 계산 속성으로 내세아세요.

리스트가 숨겨져야 할 때 렌더링을 피하고 싶을 때(예: `v-for="user in users" v-if="shouldShowUsers"`). 이 경우, `v-if` 를 컨테이너 요소(예: `ul`, `ol`)로 옮기세요.

`key` 로 상태 유지하기

Vue가 `v-for` 로 렌더링된 요소 목록을 업데이트할 때, 기본적으로 "제자리 패치(in-place patch)" 전략을 사용합니다. 데이터 항목의 순서가 변경된 경우, Vue는 DOM 요소를 항목 순서에 맞게 이동시키는 대신 각 요소를 제자리에 패치하여 해당 인덱스에 렌더링되어야 할 내용을 반영합니다.

이 기본 모드는 효율적이지만, **리스트 렌더 출력이 자식 컴포넌트 상태나 임시 DOM 상태(예: 폼 입력 값)에 의존하지 않을 때만 적합합니다**.

Vue가 각 노드의 정체성을 추적하고, 기존 요소를 재사용 및 재정렬할 수 있도록 힌트를 주려면 각 항목에 고유한 `key` 속성을 제공해야 합니다:

template

```
<div v-for="item in items" :key="item.id">
  <!-- 내용 -->
</div>
```

`<template v-for>` 를 사용할 때는 `key` 를 `<template>` 컨테이너에 지정해야 합니다:

template

```
<template v-for="todo in todos" :key="todo.name">
  <li>{{ todo.name }}</li>
</template>
```

> ⓘ 참고
>
> 여기서 `key` 는 `v-bind` 로 바인딩되는 특별한 속성입니다. 객체에 대한 `v-for` 사용에서의 속성 키 변수와 혼동하지 마세요.

가능하다면, 반복되는 DOM 콘텐츠가 단순한 경우(즉, 컴포넌트나 상태를 가진 DOM 요소가 없는 경우)를 제외하고, 또는 성능 향상을 위해 기본 동작에 의존하려는 특별한 경우가 아니라면, `v-for` 와 함께 항상 `key` 속성을 제공하는 것이 좋습니다.

요. `key` 속성의 자세한 사용법은 `key` API 문서를 참고하세요.

컴포넌트와 함께 사용하는 `v-for`

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

`v-for` 를 일반 요소처럼 컴포넌트에 직접 사용할 수 있습니다(꼭 `key` 를 제공하세요):

template
```
<MyComponent v-for="item in items" :key="item.id" />
```

하지만, 이 방법만으로는 컴포넌트에 데이터를 자동으로 전달하지 않습니다. 컴포넌트는 자체적으로 격리된 스코프를 가지기 때문입니다. 반복되는 데이터를 컴포넌트에 전달하려면 props도 함께 사용해야 합니다:

template
```
<MyComponent
  v-for="(item, index) in items"
  :item="item"
  :index="index"
  :key="item.id"
/>
```

`item` 을 컴포넌트에 자동으로 주입하지 않는 이유는, 그렇게 하면 컴포넌트가 `v-for` 의 동작 방식에 강하게 결합되기 때문입니다. 데이터의 출처를 명확히 하는 것이 컴포넌트를 다른 상황에서도 재사용할 수 있게 해줍니다.

간단한 투두 리스트 예제를 확인하여, `v-for` 로 컴포넌트 목록을 렌더링하고 각 인스턴스에 서로 다른 데이터를 전달하는 방법을 알아보세요.

배열 변경 감지

변이 메서드

Vue는 반응형 배열의 변이 메서드가 호출될 때 이를 감지하고 필요한 업데이트를 트리거할 수 있습니다. 이러한 변이 메서드는 다음과 같습니다:

```
push()
```

- `shift()`
- `unshift()`
- `splice()`
- `sort()`
- `reverse()`

배열 교체하기

변이 메서드는 이름에서 알 수 있듯이 호출된 원본 배열을 변경합니다. 반면, `filter()`, `concat()`, `slice()` 와 같은 비변이 메서드는 원본 배열을 변경하지 않고 **항상 새로운 배열을 반환**합니다. 비변이 메서드를 사용할 때는 기존 배열을 새 배열로 교체해야 합니다:

```js
// `items`는 배열 값을 가진 ref입니다
items.value = items.value.filter((item) => item.message.match(/Foo/))
```

이렇게 하면 Vue가 기존 DOM을 버리고 전체 리스트를 다시 렌더링할 것이라고 생각할 수 있지만, 다행히도 그렇지 않습니다. Vue는 DOM 요소 재사용을 극대화하기 위한 스마트한 휴리스틱을 구현하고 있으므로, 중첩된 객체가 포함된 배열을 다른 배열로 교체하는 작업도 매우 효율적으로 처리됩니다.

필터링/정렬된 결과 표시하기

때로는 원본 데이터를 실제로 변경하거나 재설정하지 않고 배열의 필터링되거나 정렬된 버전을 표시하고 싶을 때가 있습니다. 이 경우, 필터링되거나 정렬된 배열을 반환하는 계산 속성을 만들 수 있습니다.

예를 들어:

```js
const numbers = ref([1, 2, 3, 4, 5])

const evenNumbers = computed(() => {
  return numbers.value.filter((n) => n % 2 === 0)
})
```

```template
<li v-for="n in evenNumbers">{{ n }}</li>
```

계산 속성을 사용할 수 없는 상황(예: 중첩된 `v-for` 루프 내부)에서는 메서드를 사용할 수 있습니다:

```
  [1, 2, 3, 4, 5],
  [6, 7, 8, 9, 10]
])

function even(numbers) {
  return numbers.filter((number) => number % 2 === 0)
}
```

template

```
<ul v-for="numbers in sets">
  <li v-for="n in even(numbers)">{{ n }}</li>
</ul>
```

계산 속성에서 `reverse()` 와 `sort()` 를 사용할 때는 주의하세요! 이 두 메서드는 원본 배열을 변경하므로, 계산 getter에서 사용하는 것은 피해야 합니다. 이러한 메서드를 호출하기 전에 원본 배열의 복사본을 만드세요:

diff

```
- return numbers.reverse()
+ return [...numbers].reverse()
```

GitHub에서 이 페이지 편집

Previous
조건부 렌더링

Next
이벤트 처리