리스트 렌더링

v-for

v-for 디렉티브를 사용하여 배열을 리스트로 렌더링할 수 있습니다. v-for 디렉티브는 item in items 형식의 특별한 문법이 필요합니다. 여기서 items 는 배열이고, item (이하 아이템)은 배열 내 반복되는 앨리먼트의 **별칭(alias)**입니다.

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

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

v-for 범위 내 템플릿 표현식은 모든 상위 범위 속성에 접근할 수 있습니다. 또한 v-for 는 현재 아이템의 인덱스를 가리키는 선택적 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

온라인 연습장으로 실행하기

v-for 의 변수 범위는 JavaScript와 유사합니다:

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

items.forEach((item, index) => {
  // forEach의 콜백 함수 외부에 있는 `parentMessage`에 대한 접근 가능.
  // 반면 `item`과 `index`는 콜백함수 내부에서만 접근 가능.
  console.log(parentMessage, item.message, index)
})
```

v-for 에 주어진 값이 forEach 의 콜백 함수의 특징과 유사하다는 것을 눈치챘나요? 실제로 콜백 함수 인자를 분해 할당해 사용할 수 있는 것처럼, v-for 의 아이템도 분해 할당해 사용할 수 있습니다:

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

<!-- index 별칭도 사용 -->
```

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

중첩된 v-for 의 경우, 중첩된 함수와 유사한 범위를 가집니다. 각 v-for 범위에는 상위 범위에 대한 접근 권한이 있습니다.

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

in 대신 of 를 구분 기호로 사용하여 JavaScript 반복문 문법처럼 사용할 수도 있습니다:

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

객체에 v-for 사용하기

v-for 를 객체의 속성을 반복하는 데 사용할 수 있습니다. 순회 순서는 해당 객체를 Object.keys() 를 호출한 결과에 기반합니다:

```js
const myObject = reactive({
  title: 'Vue에서 목록을 작성하는 방법',
  author: '홍길동',
  publishedAt: '2016-04-10'
})
```

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

속성명을 가리키는 두 번째 별칭를 사용할 수도 있습니다:

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

그리고 인덱스를 가리키는 세 번째 별칭를 사용할 수도 있습니다:

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

온라인 연습장으로 실행하기

숫자 범위에 v-for 사용하기

v-for 는 정수를 사용할 수도 있습니다. 이 경우 1...n 범위를 기준으로 템플릿을 여러 번 반복합니다.

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

여기서 n 의 값은 0 이 아니라 1 부터 시작합니다.

<template> 에서 v-for 사용하기

v-if 와 유사하게 <template> 태그에 v-for 를 사용하여 여러 엘리먼트 블록을 렌더링할 수도 있습니다:

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

v-if 에 v-for 사용하기

참고

v-if 와 v-for 를 함께 사용하는 것은 **권장되지 않습니다.**

이것들이 같은 노드에 존재할 때 v-if 가 v-for 보다 우선순위가 높기 때문에 v-if 조건문에서 v-for 변수에 접근할 수 없습니다:

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

<template> 태그로 감싼 후, v-for 를 옮겨서 해결할 수 있습니다(더 명시적이기도 함):

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

key 를 통한 상태유지

Vue가 v-for 로 렌더링된 리스트를 업데이트할 때, 기본적으로 "in-place patch" 전략을 사용합니다. 리스트 아이템의 순서가 변경된 경우, 아이템의 순서와 일치하도록 DOM 엘리먼트를 이동하는 대신, 변경이 필요한 인덱스의 엘리먼트들을 제자리에서 패치해 아이템을 렌더링하도록 합니다.

이러한 기본동작은 효율적이지만, **리스트 렌더링 출력이 자식 컴포넌트 상태 또는 임시 DOM 상태(예: 양식 입력 값)에 의존하지 않는 경우에만 유효**합니다.

Vue가 각 노드의 ID를 추적하고 기존 엘리먼트를 재사용하고 재정렬할 수 있도록 힌트를 제공하려면 각 항목에 대해 고유한 key 속성을 제공해야 합니다.

> 역자주
>
> DOM Node의 위치가 변경되면 DOM Tree구조가 변경 되었기 때문에 Render Tree를 재구성한후에, 재 구성된 Node들의 위치,크기,깊이등을 다시 계산하는 Reflow 과정을 수행하고, DOM Node를 다시 그려 내는 Repaint 과정을 거치게 됩니다. 만약 Node를 이동하지 않고 Node 내부의 컨텐츠만 변경한다면, 렌더 트리 재구성이나 Reflow를 생략하고 해당 Node를 Repaint하기 때문에 효율적입니다.
>
> 만약 자식 컴포넌트가 상태를 가진다면(예를 들어, Input 폼에 사용자가 값을 입력 해두었다면, 그 입력값을 지우면 안되기 때문에) 해당 자식 컴포넌트를 재사용 할 수 없기 때문에, 이렇게 기존 DOM Node를 재사용하는 방법은 유효한 방법이 아닐 것 입니다.
>
> 참고: Render-tree Construction, Layout, and Paint

template

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

<template v-for> 를 사용할 때 key 는 <template> 컨테이너에 있어야 합니다.

template

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

> 참고
>
> 여기서 key 는 v-bind 와 결합되는 특수 속성입니다. 객체에 v-for 사용하기에서 언급하는 두 번째 별칭인 key와 혼동해서는 안 됩니다.

반복되는 DOM 컨텐츠가 단순하거나(컴포넌트도 없고, 상태를 가지는 DOM 앨리먼트도 없을때), 의도적으로 기본 리스트 렌더링 동작을 통해 성능 향상을 꾀하는 경우가 아니라면, 가능한 한 언제나 v-for 는 key 속성과 함께 사용하는 것을 권장합니다.

key 에는 문자열, 숫자, 심볼 형식의 값만 바인딩해야 합니다. key 속성의 자세한 사용법은 key API 문서를 참조하세요.

컴포넌트에 v-for 사용하기

이 섹션은 컴포넌트에 대한 지식이 있다고 가정하므로, 건너뛰고 나중에 읽어도 됩니다.

컴포넌트에 v-for 를 직접 사용할 수 있습니다(key 사용을 잊지 마세요):

template

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

그러나 컴포넌트에는 자체적으로 구분된 범위가 있기 때문에 컴포넌트에 데이터를 자동으로 전달하지 않습니다. 반복된 데이터를 컴포넌트에 전달하려면 props를 사용해야 합니다.

template

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

컴포넌트에 item 이 자동으로 전달되지 않는 이유는, 그렇지 않으면 v-for 를 사용해야만 컴포넌트 사용이 가능하도록 의존관계가 되기 때문입니다. 반면, 데이터를 명시해서 전달하는 방법은 v-for 를 사용하지 않는 다른 상황에서도 컴포넌트가 데이터 전달 기능을 재사용할 수 있기 때문입니다.

각 인스턴스에 다른 데이터를 전달하는 v-for 를 사용하여 컴포넌트 리스트를 렌더링하는 방법을 보려면 간단한 할 일 목록 예제를 확인하세요.

배열 변경 감지

수정 메서드

Vue는 반응형 배열의 변경 메소드가 호출 되는것을 감지하여, 필요한 업데이트를 발생시킵니다. 지원하는 변경 메소드 목록은 다음과 같습니다:

- push()
- pop()
- shift()
- unshift()
- splice()
- sort()
- reverse()

배열 교체

수정 메서드는 이름에서 알 수 있듯이 호출된 원래 배열을 수정합니다. 이에 비해 수정이 아닌 방법도 있습니다. filter() , concat() 및 slice() 는 원본 배열을 수정하지 않고 **항상 새 배열을 반환합니다**. 이러한 방법으로 작업하는 경우, 이전 배열을 새 배열로 교체해야 합니다.

js

```
// `items`는 값이 있는 배열의 ref라고 가정된 경우입니다.
items.value = items.value.filter((item) => item.message.match(/Foo/))
```

이로 인해 Vue가 기존 DOM을 버리고 전체 리스트를 다시 렌더링할 것이라고 생각할 수도 있습니다. 다행히도 그렇지 않습니다. Vue는 DOM 엘리먼트 재사용을 최대화하기 위해 몇 가지 스마트 휴리스틱을 구현하므로, 이전 배열을 다른 배열로 바꾸는 과정에서 서로 중복되는 객체를 가지는 부분을 매우 효율적으로 처리합니다.

필터링/정렬 결과 표시

때로는 원본 데이터를 실제로 수정하거나 교체하지 않고 필터링되거나 정렬된 결과를 표시하고 싶을 수 있습니다. 이 경우 필터링되거나 정렬된 배열을 반환하는 계산된 속성을 만들 수 있습니다.

예제 코드:

```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>
```

계산된 속성이 실현 가능하지 않은 상황(예: 중첩된 `v-for` 루프 내부)에서는 다음과 같은 방법으로 해결할 수 있습니다:

```js
const sets = ref([
  [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>
```

계산된 속성에서 `reverse()` 와 `sort()` 사용에 주의하십시오! 이 두 가지 방법은 원본 배열을 수정하므로 계산된 속성의 getter 함수에서 피해야합니다. 다음 메서드를 호출하기 전에 원본 배열의 복사본을 만듭니다:

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