리스트 렌더링

v-for

`v-for` 디렉티브를 사용하여 일반 배열, 객체 배열 등 리스트 렌더링을 할 수 있다

```
<tr v-for="fruit in fruits">
  {{ fruit }}
</tr>
```

숫자 범위에 활용

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

v-if와 v-for

`v-if`와 `v-for`를 함께 사용할 수 있지만 권장되지 않는다.

`v-if`가 `v-for`보다 우선순위가 높기 때문에 `v-if`에서 `v-for`변수에 접근할 수 없다.

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

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

`key`를 통한 상태유지

Vue가 `v-for` 로 렌더링 된 리스트를 업데이트할 때, 기본적으로 **in-place patch** 전략을 사용한다. 리스트 아이템의 순서가 변경된 경우, 아이템의 순서와 일치하도록 DOM 엘리먼트를 이동하는 대신, 변경이 필요한 인덱스의 엘리먼트들을 제자리에서 패치(patch)해 아이템을 렌더링 하도록 한다.

이런 기본 동작은 효율적이지만, 리스트 렌더링 출력이 자식 컴포넌트 상태 또는 임시 DOM 상태 (ex: 양식 입력 값)에 의존하지 않는 경우에만 유효하다.

```
This default mode is efficient, but only suitable when your list render output does not rely on child component state or temporary DOM state (e.g. form input values).
```

javascript에서 Temporary DOM 이란?

Javascript에서 Temporary DOM이란, 웹 페이지의 Document Object Model(DOM)에서 일시적으로 생성된 요소의 집합을 의미합니다.

일반적으로 Javascript를 사용하여 DOM을 조작하면, 새로운 HTML 요소를 동적으로 생성하고 기존 요소를 수정, 삭제할 수 있습니다. 이 때, Temporary DOM은 이러한 동적인 변경 작업을 수행하기 위해 임시적으로 생성되는 DOM 구조입니다.

Temporary DOM은 일반적으로 메모리에 생성되며, 웹 페이지의 실제 DOM에는 영향을 미치지 않습니다. 이는 작업 중에 일부 변경 사항을 롤백할 수 있게 해주는데 유용합니다.

예를 들어, 새로운 HTML 요소를 생성하고 DOM에 추가한 후, 일부 오류가 발생하여 요소를 제거해야 할 경우, Temporary DOM을 사용하여 요소를 제거하고 DOM을 롤백할 수 있습니다. 이렇게 하면 오류가 발생하기 전 상태로 DOM을 되돌릴 수 있습니다.

`!(자식 컴포넌트에 의존) && 임시 DOM 상태`

Default

```
<template>
<div>
  <h1>v-bind:key="None"</h1>
  <table>
    <thead>
      <tr>
        <th>None</th>
        <th>Name</th>
        <th>Value</th>
      </tr>
    </thead>
    <tbody>
      <tr v-for="fruit in fruits0"> <!-- eslint 빨간줄 -->
        <td> - </td>
        <td>{{ fruit.name }}</td>
        <td>
          <input type="number" />
        </td>
      </tr>
    </tbody>
  </table>
  <button v-on:click="shift0">shift</button>
</div>
<template>

<script setup>
  import { ref } from "vue";

  const fruits0 = ref([
    { id: 1, name: "apple" },
    { id: 2, name: "orange" },
    { id: 3, name: "melon" },
    { id: 4, name: "grape" },
    { id: 5, name: "mango" }
  ]);

  const shift0 = () => fruits0.value.push(fruits0.value.shift());
</script>
```

Before	After
v-bind:key="Array Index"	v-bind:key="Array Index"

Index	Name	Value
0	apple	1
1	orange	2
2	melon	3
3	grape	4
4	mango	5

shift

Index	Name	Value
0	melon	1
1	grape	2
2	mango	3
3	apple	4
4	orange	5

shift

Array index key

```
<template>
<div>
  <h1>v-bind:key="Array Index"</h1>
  <table>
    <thead>
      <tr>
        <th>Index</th>
        <th>Name</th>
        <th>Value</th>
      </tr>
    </thead>
    <tbody>
      <tr v-for="(fruit, i) in fruits1" :key="i">
        <td>{{ i }}</td>
        <td>{{ fruit.name }}</td>
        <td>
          <input type="number" />
        </td>
      </tr>
    </tbody>
  </table>
  <button v-on:click="shift1">shift</button>
</div>
</template>
<script setup>
  import { ref } from "vue";

  const fruits1 = ref([
    { id: 1, name: "apple" },
    { id: 2, name: "orange" },
    { id: 3, name: "melon" },
    { id: 4, name: "grape" },
    { id: 5, name: "mango" }
  ]);

  const shift1 = () => fruits1.value.push(fruits1.value.shift());
</script>
```

Before	After
v-bind:key="None"	v-bind:key="None"

None	Name	Value
-	apple	1
-	orange	2
-	melon	3
-	grape	4
-	mango	5

shift

None	Name	Value
-	melon	1
-	grape	2
-	mango	3
-	apple	4
-	orange	5

shift

0번 인덱스에서 `fruit` 객체가 `shift` 되지만, 다시 0번 인덱스가 바로 채워진다.

또한, 4번 인덱스가 3번으로 당겨지지만, 바로 4번 인덱스에 값이 채워진다.

따라서 키가 변하지 않으므로 `tr` 객체에는 변화가 없다. 그래서 `input` 태그는 변화가 없고, 내부의 `{{ fruit.name }}` 만 변경된다.

Object Id key

```
<template>
<div>
  <h1>v-bind:key="Object Id"</h1>
  <table>
    <thead>
      <tr>
        <th>Object Id</th>
        <th>Name</th>
        <th>Value</th>
      </tr>
    </thead>
    <tbody>
      <tr v-for="fruit in fruits2" :key="fruit.id">
        <td>{{ fruit.id }}</td>
        <td>{{ fruit.name }}</td>
        <td><input type="number" /></td>
      </tr>
    </tbody>
  </table>
  <button v-on:click="shift2">shift</button>
</div>
</template>
<script setup>
  import { ref } from "vue";

  const fruits2 = ref([
    { id: 1, name: "apple" },
    { id: 2, name: "orange" },
    { id: 3, name: "melon" },
    { id: 4, name: "grape" },
    { id: 5, name: "mango" }
  ]);

  const shift2 = () => fruits2.value.push(fruits2.value.shift());
</script>
```

`fruit.id`라는 고유값을 가진 `tr` 태그가 `shift` 후 다른 `fruit.id`를 가진 `tr` 태그로 채워지므로 `tr` 태그 자체가 다른 값으로 바뀌었다.
따라서 `input` 태그 또한 함께 이동하였다.

Before	After
v-bind:key="Object Id"	v-bind:key="Object Id"

Object Id	Name	Value
a	apple	1
b	orange	2
c	melon	3
d	grape	4
e	mango	5

shift

Object Id	Name	Value
c	melon	3
d	grape	4
e	mango	5
a	apple	1
b	orange	2

shift

결론

반복되는 DOM 컨텐츠가 단순하거나, 의도적으로 렌더링 동작을 통해 성능 향상을 꾀하는 경우가 아니라면, 가능한 `v-for` 는 `key` 속성과 함께 사용하는 것을 권장한다.

```
key: number | string | symbol
```

애니메이션의 객체 불변성이란?

애니메이션의 객체 불변성(Immutability of Objects in Animations)이란, 애니메이션 프레임 간에 객체의 속성 값이 변경되지 않고 유지되는 것을 의미합니다.

일반적으로, 애니메이션에서는 객체의 위치, 크기, 색상 등의 속성을 변화시켜 애니메이션을 구현합니다. 이때, 객체의 속성 값이 변경되면서 새로운 객체가 생성되고, 기존 객체는 삭제됩니다. 이러한 변경 작업은 성능 저하와 메모리 누수를 야기할 수 있습니다.

객체 불변성을 유지하면, 새로운 객체를 생성하지 않고도 속성 값을 업데이트할 수 있습니다. 예를 들어, 애니메이션에서 객체의 위치를 변경할 때, 기존 객체를 삭제하고 새로운 객체를 생성하는 대신, 기존 객체의 위치 속성 값을 변경하는 것입니다.

객체 불변성은 성능과 메모리 사용량을 최적화하면서도 애니메이션을 자연스럽게 보여줄 수 있습니다. 이는 React나 Vue.js와 같은 UI 라이브러리에서도 중요한 개념으로 사용됩니다.

배열 변경 감지

수정 메서드

- Vue는 반응형 배열의 메서드가 호출되는 것을 감지하여, 필요한 업데이트를 발생시킵니다.

`push()` | `pop()` | `shift()` `unshift()` | `splice()` | `sort()` | `reverse()`

배열 교체

- `filter()` | `concat()` | `slice()` 는 원본 배열을 수정하지 않고 새 배열을 반환하므로 기존의 배열(ref([]). value)을 변경해야 한다.

감사합니다

출처

- https://ko.vuejs.org/guide/essentials/list.html
- https://goodteacher.tistory.com/525