Renderizado de lista

Mapeando una matriz a elementos con `v-for`

Podemos usar la directiva `v-for` para representar una lista de elementos basada en una matriz. La directiva `v-for` requiere una sintaxis especial en forma de `item in items`, donde los `items` son la matriz de datos de origen y el `item` es un **alias** para el elemento de matriz que se está iterando:

HTML

```
<ul id="example-1">
  <li v-for="item in items">
    {{ item.mensaje }}
  </li>
</ul>
```

JS

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    items: [
      { mensaje: 'Foo' },
      { mensaje: 'Bar' }
    ]
  }
})
```

Resultado:

- Foo
- Bar

Dentro de los bloques `v-for` tenemos acceso completo a las propiedades del ámbito principal. `v-for` también admite un segundo argumento opcional para el índice del elemento actual.

HTML

```
<ul id="example-2">
  <li v-for="(item, index) in items">
    {{ MensajePadre }} - {{ index }} - {{ item.mensaje }}
  </li>
</ul>
```

JS

```
var example2 = new Vue({
  el: '#example-2',
```

```
    items: [
      { mensaje: 'Foo' },
      { mensaje: 'Bar' }
    ]
  }
})
```

Resultado:

- Padre - 0 - Foo
- Padre - 1 - Bar

También puede usar `of` como delimitador en lugar de `in`, de modo que esté más cerca de la sintaxis de JavaScript para los iteradores:

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

`v-for` con un Objeto

También puede usar `v-for` para iterar a través de las propiedades de un objeto.

HTML
```
<ul id="v-for-object" class="demo">
  <li v-for="value in object">
    {{ value }}
  </li>
</ul>
```

JS
```
new Vue({
  el: '#v-for-object',
  data: {
    object: {
      primerNombre: 'John',
      apellido: 'Doe',
      edad: 30
    }
  }
})
```

Resultado:

- John
- Doe

También puede proporcionar un segundo argumento para la clave:

```html
<div v-for="(value, key) in object">
  {{ key }}: {{ value }}
</div>
```

primerNombre: John
apellido: Doe
edad: 30

Y otro para el índice:

```html
<div v-for="(value, key, index) in object">
  {{ index }}. {{ key }}: {{ value }}
</div>
```

0. primerNombre: John
1. apellido: Doe
2. edad: 30

! Al iterar sobre un objeto, el orden se basa en el orden de enumeración de claves de `Object.keys()`, que no se garantiza que sea consistente en todas las implementaciones del motor de JavaScript.

key

Cuando Vue está actualizando una lista de elementos representados con `v-for`, por defecto utiliza una estrategia de “parche in situ”. Si el orden de los elementos de datos ha cambiado, en lugar de mover los elementos DOM para que coincidan con el orden de los elementos, Vue aplicará parches a cada elemento en el lugar y se asegurará de que refleje lo que se debe representar en ese índice en particular. Esto es similar al comportamiento de `track-by="$index"` en Vue 1.x.

ejemplo, valores de entrada de formulario).

Para proporcionar a Vue una sugerencia para que pueda rastrear la identidad de cada nodo y, por lo tanto, reutilizar y reordenar los elementos existentes, debe proporcionar un atributo `key` único para cada elemento. Un valor ideal para `key` sería el ID único de cada elemento. Este atributo especial es un equivalente aproximado a `track-by` en 1.x, pero funciona como un atributo, por lo que necesita usar `v-bind` para enlazarlo con valores dinámicos (usando el modo abreviado aquí):

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

Se recomienda proporcionar una `key` con `v-for` siempre que sea posible, a menos que el contenido DOM iterado sea simple, o esté confiando intencionalmente en el comportamiento predeterminado para obtener ganancias en el rendimiento.

Como Vue es un mecanismo genérico para identificar nodos, la `key` también tiene otros usos que no están específicamente vinculados a `v-for`, como veremos más adelante en la guía.

Deteccion del cambios en Array

Metodos de Mutacion

Vue envuelve los métodos de mutación de una matriz observada para que también activen las actualizaciones de vista. Los métodos envueltos son:

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

Puede abrir la consola y probar con la matriz de `items` de los ejemplos anteriores llamando a sus métodos de mutación. Por ejemplo: `example1.items.push ({mensaje: 'Baz'})` .

Mutando un Array

no mutan la matriz original pero **siempre devuelven una nueva matriz**. Cuando trabaje con métodos no mutantes, puede reemplazar la matriz anterior por la nueva:

```js
example1.items = example1.items.filter(function (item) {
  return item.mensaje.match(/Foo/)
})
```

Podría pensar que esto hará que Vue elimine el DOM existente y vuelva a renderizar la lista completa; afortunadamente, ese no es el caso. Vue implementa algunas heurísticas inteligentes para maximizar la reutilización de elementos DOM, por lo tanto, reemplazar una matriz con otra matriz que contenga objetos superpuestos es una operación muy eficiente.

Advertencias

Debido a las limitaciones en JavaScript, Vue **no puede** detectar los siguientes cambios en una matriz:

1. Cuando configura directamente un elemento con el índice, por ejemplo, `vm.items[indexOfItem] = newValue`
2. Cuando modifica la longitud de la matriz, por ejemplo, `vm.items.length = newLength`

Por ejemplo:

```js
var vm = new Vue({
  data: {
    items: ['a', 'b', 'c']
  }
})
vm.items[1] = 'x' // NO es reactivo
vm.items.length = 2 //  NO es reactivo
```

Para superar la advertencia 1, ambos de los siguientes lograrán lo mismo que `vm.items[indexOfItem] = newValue` , pero también activarán actualizaciones de estado en el sistema de reactividad:

```js
// Vue.set
Vue.set(vm.items, indexOfItem, newValue)
```

```js
// Array.prototype.splice
vm.items.splice(indexOfItem, 1, newValue)
```

También puede usar el método de instancia `vm.$Set` , que es un alias para el `Vue.set` global:

Para tratar con la advertencia 2, puede usar `splice` :

```
JS
vm.items.splice(newLength)
```

Advertencias con la Deteccion de Cambios en Objetos

Una vez más, debido a las limitaciones del JavaScript moderno, **Vue no puede detectar la adición o eliminación de propiedades**. Por ejemplo:

```
JS
var vm = new Vue({
  data: {
    a: 1
  }
})
// `vm.a` ahora es reactivo

vm.b = 2
// `vm.b` NO es reactivo
/
```

Vue no permite agregar dinámicamente nuevas propiedades reactivas a nivel de raíz a una instancia ya creada. Sin embargo, es posible agregar propiedades reactivas a un objeto anidado usando el método `Vue.set (objeto, clave, valor)` . Por ejemplo, dado:

```
JS
var vm = new Vue({
  data: {
    userProfile: {
      name: 'Anika'
    }
  }
})
```

Podría agregar una nueva propiedad de `edad` al objeto de `userProfile` anidado con:

```
JS
Vue.set(vm.userProfile, 'edad', 27)
```

También puede usar el método de instancia vm. `$Set` , que es un alias para el `Vue.set` global:

```
JS
vm.$set(vm.userProfile, 'edad', 27)
```

En ocasiones, es posible que desee asignar varias propiedades nuevas a un objeto existente, por ejemplo, utilizando `Object.assign()` o `_.extend()` . En tales casos, debe crear un objeto nuevo con propiedades de ambos objetos. Así que en lugar de:

```
  edad: 27,
  favoriteColor: 'Vue Green'
})
```

Puedria agregar nuevas propiedades reactivas con:

```JS
vm.userProfile = Object.assign({}, vm.userProfile, {
  edad: 27,
  favoriteColor: 'Vue Green'
})
```

Mostrando Resultados Filtrados/Ordenados

A veces, queremos mostrar una versión filtrada u ordenada de una matriz sin mutar o restablecer los datos originales. En este caso, puede crear una propiedad computada que devuelva la matriz filtrada o ordenada.

Por ejemplo:

```HTML
<li v-for="n in numerosImpares">{{ n }}</li>
```

```JS
data: {
  numeros: [ 1, 2, 3, 4, 5 ]
},
computed: {
  numerosImpares: function () {
    return this.numeros.filter(function (numero) {
      return numero % 2 === 0
    })
  }
}
```

En situaciones donde las propiedades computadas no son factibles (por ejemplo, dentro de los bucles `v-for` anidados), puede usar un método:

```HTML
<li v-for="n in even(numeros)">{{ n }}</li>
```

```JS
data: {
  numeros: [ 1, 2, 3, 4, 5 ]
},
methods: {
  even: function (numeros) {
    return numeros.filter(function (numero) {
      return numero % 2 === 0
    })
  }
}
```

`v-for` con un Rango

`v-for` también puede tomar un entero. En este caso repetirá la plantilla muchas veces.

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

Resultado:

```
1 2 3 4 5 6 7 8 9 10
```

`v-for` en un `<template>`

De forma similar a la plantilla `v-if`, también puede usar una etiqueta `<template>` con `v-for` para renderizar un bloque de varios elementos. Por ejemplo:

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

`v-for` con `v-if`

> ! Tenga en cuenta que **no se recomienda** usar `v-if` y `v-for` juntos. Consulte la **guía de estilo** para más detalles.

Cuando existen en el mismo nodo, `v-for` tiene una prioridad más alta que `v-if`. Eso significa que el `v-if` se ejecutará en cada iteración del bucle por separado. Esto puede ser útil cuando desea representar nodos solo para *algunos* elementos, como a continuación:

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

Si, por el contrario, su intención es omitir condicionalmente la ejecución del bucle, puede colocar el `v-if` en un elemento de envoltura (o `<template>`). Por ejemplo:

```HTML
<ul v-if="todos.length">
  <li v-for="todo in todos">
    {{ todo }}
  </li>
</ul>
<p v-else>No quedan todos !</p>
```

`v-for` con un Componente

> **Esta sección asume el conocimiento de Componentes. Siéntase libre de saltearlo y volver más tarde.**

Puede usar `v-for` directamente en un componente personalizado, como cualquier elemento normal:

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

> **En 2.2.0+, cuando se usa `v-for` con un componente, ahora se requiere una `key` .**

Sin embargo, esto no pasará automáticamente ningún dato al componente, porque los componentes tienen sus propios ámbitos aislados. Para pasar los datos iterados al componente, también debemos usar props:

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

La razón para no inyectar automáticamente el `item` en el componente es porque hace que el componente esté estrechamente acoplado a cómo funciona `v-for` . Ser explícito acerca de dónde provienen sus datos hace que el componente sea reutilizable en otras situaciones.

Aquí hay un ejemplo completo de una lista de tareas simple:

```HTML
<div id="todo-list-example">
  <form -on:submit.prevent="addNewTodo">
    <label for="new-todo">Agregar tarea</label>
    <input
      v-model="newTodoText"
```

```
    <button>Agregar</button>
  </form>
  <ul>
    <li
      is="todo-item"
      v-for="(todo, index) in todos"
      v-bind:key="todo.id"
      v-bind:title="todo.title"
      v-on:remove="todos.splice(index, 1)"
    ></li>
  </ul></div>
```

! Note el atributo `is = "todo-item"` . Esto es necesario en las plantillas DOM, porque solo un elemento `<li>` es válido dentro de un `<ul>` . Hace lo mismo que `<todo-item>` , pero funciona alrededor de un error potencial de análisis del navegador. Ver las **advertencias de análisis de plantillas DOM** aprender más.

JS

```
.component('todo-item', {
  template: '\
    <li>\
      {{ title }}\
      <button -on:click="$emit(\'remove\')">Remove</button>\
    </li>\
  ',
  props: ['title']
})

new Vue({
  el: '#todo-list-example',
  data: {
    newTodoText: '',
    todos: [
      {
        id: 1,
        title: 'Do the dishes'
      },
      {
        id: 2,
        title: 'Take out the trash'
      },
      {
        id: 3,
        title: 'Mow the lawn'
      }
    ],
    nextTodoId: 4
  },
  methods: {
    addNewTodo: function () {
      this.todos.push({
        id: this.nextTodoId++,
        title: this.newTodoText
      })
      this.newTodoText = ''
    }
```

Agregar una tarea E.g. Alimentar al gato Agregar

- Lavar los platos Eliminar
- Sacar la basura Eliminar
- Cortar el césped Eliminar

← Renderización Condicional

Manejo de eventos →