Build an Online Shop with Vue: Lesson 8 of 33

Conditional & List Rendering

NEXT: STYLES & CLASSES

Still on templates -- apart from inline expressions, we could also use basic flow controls like `if...else` and `for` statements. This is no way a native JavaScript feature but just syntactic sugar from Vue to make conditional and list rendering easy.

In this section, we will review these Vue features, compare them with native JS implementation and see some examples.

Conditional Rendering

There are situations where you would want to dynamically hide or show a particular item in the DOM based on the value of some variable. This variable is most times a boolean flag. With vanilla JavaScript, we could do this:

Table of Contents

```html
<!-- Click #btn to hide #text -->
<button id="btn">Clicke me</button>
<p id="text">Lorem ipsum dolor sit amet,...</p>

<script>
    let isShown = false;

    const btn = document.getElementById('btn');
    const text = document.getElementById('text');

    updateText(isShown)

    btn.addEventListener('click', () => {
        isShown = !isShown
        console.log(isShown)
        updateText(isShown)
    })

    function updateText(isShown) {
        text.style.display = isShown ? 'block' :'none';
    }
</script>
```

No doubt. The snippet is sufficient enough to toggle a text when a button is clicked by manipulating the text's style.

Now let's compare the above with how we can achieve this toggle behavior in Vue:

```html
<template>
    <!--
    Click to invoke toggle method which
    toggles the boolean flag
    -->
    <button @click="toggle">Click me</button>

    <!-- Show if 'isShown' is true -->
    <p id="text" v-show="isShown">Lorem ipsum dolor sit amet,...</p>
</template>

<script>
export default {
    data() {
        return {
            isShown: false, // Control property
        };
    },
    methods: {
        toggle() {
        // Toggles control property
        this.isShown = !this.isShown;
        },
    },
}
</script>
```

The most important aspect is using the `v-show` attribute to show or hide the paragraph tag whenever `isShown` property is `true`.

The `v-show` conditional rendering directive is efficient when you need to hide an element that could be shown later like the above toggle example.

There is another variation of conditional rendering which is `if...else`. This variation is more effective when you are certain that an element will be hidden throughout the current state of an application. This is because the DOM item is completely removed from the DOM tree and not hidden with CSS. A good example is displaying user authentication status:

```html
<template>
    <!-- Is authenticated -->
    <p id="text" v-if="isAuth">Welcome...</p>

    <!-- Is NOT authenticated-->
    <button else>Login</button>
</template>

<script>
  export default {
      data() {
            return {
                // Control property
                isAuth: false,
            };
        },
        created: {
            // Attempt authenication
            performAuth().then(auth => this.isAuth = auth.username ? true : false)
        },
  }
</script>
```

List Rendering

When you have an array of data that needs to be iteratively rendered on the browser. The canonical way was to create a list of DOM elements with each of the elements matching each item in the data array. This list of DOM elements is then *appended* to their parent element. This is tedious and most times gets out of control.

Rather than this manual DOM iteration, Vue allows you to loop an array of data right inside the template:

```html
<template>
    <ul>
        <li for="product in products">{{product.name}}</p>
    </ul>
</template>

<script>
    export default {
        data() {
            return {
                products: [
                    {id: 1, name: 'iPhone7'}
                    {id: 2, name: 'iPhone6'}
                ]
            }
        },
    }
</script>
```

Conditional Groups

In the previous sections, our conditions and loops contained a single line element:

```html
<!-- v-show -->
<p id="text" v-show="isShown">Lorem ipsum dolor sit amet,...</p>

<!-- v-if -->
<p id="text" v-if="isAuth">Welcome...</p>

<!-- v-for -->
<li for="product in products">{{product.name}}</p>
```

This might not always be the case. In some situations, you might need to have a group of nested elements lives in another element that contains one of these directives. Example:

```html
<div class="card" v-for="product in products">
  <div class="card-header">{{product.header}}</div>
  <div class="card-content">{{product.content}}</div>
  <div class="card-footer">{{product.created_at}}</div>
</div>
```

The above example is what your instincts may attempt. Vue doesn't work this way. Rather, you could wrap the template in a `<template>` tag like so:

```html
<template v-for="product in products">
    <div class="card">
        <div class="card-header">{{product.header}}</div>
        <div class="card-content">{{product.content}}</div>
        <div class="card-footer">{{product.created_at}}</div>
    </div>
</template>
```

Same thing applies to conditional rendering:

```html
<template v-if="isShown">
    <div class="card">
        <div class="card-header">{{product.header}}</div>
        <div class="card-content">{{product.content}}</div>
        <div class="card-footer">{{product.created_at}}</div>
    </div>
</template>
```

Iteration Key

Vue uses an in-place patch strategy to update items in a rendered list. Therefore, rather than recreate this list, Vue finds the particular item than changed and updates it. This is way much performant than recreating and re-rendering the list.

To make this patching easier for Vue, it is recommended that you provided a unique identifier for each node in the list using the `key` directive:

```html
<template v-for="product in products" :key="product.id">
    <div class="card">
        <div class="card-header">{{product.header}}</div>
        <div class="card-content">{{product.content}}</div>
        <div class="card-footer">{{product.created_at}}</div>
    </div>
</template>
```