Vue.js Computed Property

Topics : Vue
Written on January 11, 2024

In Vue.js, computed properties are used to define reactive data based on other reactive data in your component. Computed properties are cached and only re-evaluated when one of their dependencies changes. They are particularly useful for complex logic or calculations that depend on data in your component.

Here's an example of using a computed property in a Vue.js component:

```
<template>
<div>
<p>Original message: {{ message }}</p>
<p>Computed reversed message: {{ reversedMessage }}</p>
</div>
</template>

<script>
export default {
data() {
return {
message: 'Hello Vue.js!'
};
},
computed: {
reversedMessage() {
// This computed property depends on the 'message' data property
// It will be re-evaluated whenever 'message' changes
return this.message.split('').reverse().join('');
}
}
};
</script>
```

In this example:

- The `message` data property holds a simple string.
- The `reversedMessage` computed property depends on the `message` property and returns the reversed version of it.

You can access computed properties in your template just like regular data properties, and Vue.js will take care of keeping them up-to-date when their dependencies change.

Computed properties are also useful for more complex scenarios, such as filtering and sorting lists, performing calculations based on user input, and more.

Here's another example with a computed property that filters an array:

```
<template>
<div>
<ul>
<li v-for="evenNumber in evenNumbers">{{ evenNumber }}</li>
</ul>
</div>
</template>

<script>
export default {
data() {
return {
numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
};
},
computed: {
evenNumbers() {
// This computed property filters even numbers from the 'numbers' array
return this.numbers.filter(number => number % 2 === 0);
}
}
};
</script>
```

In this example, the `evenNumbers` computed property filters out the even numbers from the `numbers` array, providing a new array that only contains even numbers.