COMPUTED TRONG VUE 3

NỘI DUNG CHÍNH

I GIỚI THIỆU VỀ COMPUTED

II SO SÁNH COMPUTED VÀ METHOD

III WATCHER TRONG VUE.JS

1. Computed

- 1. Computed là gì:
- **Computed property trong Vue.js** thể hiện dưới dạng một phương thức hoặc một đối tượng (object) chứa các phương thức setter và getter dùng để thiết lập dữ liệu và lấy dữ liệu đưa ra template. Khi gọi một computed property thì gọi nó như một thuộc tính bình thường. Và khi đó các hành động trong computed property sẽ được thực thi. Ngoài ra chúng ta cũng có thể cho rằng **Computed Properties** là những hàm (methods) sử dụng để xử lý dữ liệu hiển thị lên template nhưng tối ưu hơn methods kết quả của nó sẽ được lưu trữ (Cache) và chỉ cập nhật khi cần thiết.

1. Computed

- Computed Properties trong Vue 3 cho phép tạo ra các giá trị được tính toán dựa trên các biến phản ứng (reactive data).
- Các giá trị này được tự động cập nhật khi các biến liên quan thay đổi.
- **Lợi ích:**

✓ Tự động theo dõi phụ thuộc: **Computed properties** tự động theo dõi các biến phản ứng liên quan và cập nhật khi có thay đổi.

✓ Tối ưu hóa hiệu suất: Kết quả được lưu trữ và chỉ tái tính toán khi các biến phụ thuộc thay đổi, tránh việc tính toán lại không cần thiết.

✓ Dễ đọc và bảo trì: Tách biệt logic phức tạp khỏi template, giúp mã nguồn gọn gàng hơn.

1. Computed

- 2. Ví dụ Vue 2:

Muốn hiển thị các thông báo khác nhau tùy thuộc vào việc **author** đã có sách hay chưa

```
Vue.createApp({
  data() {
    return {
      author: {
        name: 'John Doe',
        books: [
          'Vue 2 - Advanced Guide',
          'Vue 3 - Basic Guide',
          'Vue 4 - The Mystery'
        ]
      }
    }
  }
})
```

```
<div id="computed-basics">
  <p>Has published books:</p>
  <span>{{ author.books.length > 0 ? 'Yes' : 'No' }}</span>
</div>
```

1. Computed

- 2. Ví dụ Vue 3:

```
<script setup>
import { reactive, computed } from 'vue'

const author = reactive({
  name: 'John Doe',
  books: [
    'Vue 2 - Advanced Guide',
    'Vue 3 - Basic Guide',
    'Vue 4 - The Mystery'
  ]
})

// a computed ref
const publishedBooksMessage = computed(() => {
  return author.books.length > 0 ? 'Yes' : 'No'
})
</script>

<template>
  <p>Has published books:</p>
  <span>{{ publishedBooksMessage }}</span>
</template>
```

publishedBooksMessage là một computed property, tự động cập nhật khi số lượng sách của tác giả thay đổi.

1. Computed

- 2. Ví dụ:

Ví dụ trên nó thực hiện phép tính phụ thuộc vào **author.books**. Vấn đề này sẽ phức tạp hơn nếu chúng ta muốn đưa nó vào phép tính của template.

Đó là lý do tại sao đối với những phép tính logic phức tạp phải dụng **computed**

```
<div id="computed-basics">
  <p>Has published books:</p>
  <span>{{ publishedBooksMessage }}</span>
</div>
```

1. Computed

- 2. Ví dụ:

```
Vue.createApp({
  data() {
    return {
      author: {
        name: 'John Doe',
        books: [
          'Vue 2 - Advanced Guide',
          'Vue 3 - Basic Guide',
          'Vue 4 - The Mystery'
        ]
      }
    }
  },
  computed: {
    // a computed getter
    publishedBooksMessage() {
      // `this` points to the vm instance
      return this.author.books.length > 0 ? 'Yes' : 'No'
    }
  }
}).mount('#computed-basics')
```

Result

Has published books:

Yes

1. Computed

- Có thể liên kết dữ liệu với các thuộc tính được tính toán trong các mẫu giống như một thuộc tính bình thường. Vue biết rằng vm.publishedBooksMessage phụ thuộc vào vm.author.books, vì vậy nó sẽ cập nhật bất kỳ ràng buộc nào phụ thuộc vào vm.publishedBooksMessage khi vm.author.books thay đổi.
- **Cú pháp Computed property**

➢ Cú pháp để khai báo computed property trong Vue.js khá đơn giản chúng ta chỉ cần đặt nó ở trong key computed

```
computed : {
    convertToUpper: function (){
        return this.name.toUpperCase();
    }
}
```

2. Computed và Method

- **Computed** được khai báo như hàm (**methods**), tuy nhiên nó được lưu vào đối tượng Vue và hàm này được thực thi khi có bất kỳ dữ liệu nào bên trong nó thay đổi, làm cho tốc độ chương trình tăng hiệu suất cao.
- Đối với **methods** vì nó là lời gọi hàm nên khi có lời gọi đến hàm đó thì nó luôn luôn được gọi, do đó nó tính toán dữ liệu không dành riêng cho data, computed tính toán dựa trên dữ liệu có sẵn trong Vue nên có thể ràng buộc dữ liệu như những thuộc tính khác giúp dễ kiểm tra

2. Computed và Method

- Khi gọi **computed** ta không được thêm cặp dấu () đằng sau, điều đó tức là **computed** cũng không thể nhận tham số đầu vào như **methods**

- Vì không nhận tham số đầu vào nên **computed** chỉ nên dùng với các dữ liệu có sẵn trong data của component

- **computed** sẽ chỉ tính toán lại mỗi khi các biến phụ thuộc trong nó thay đổi, còn methods sẽ được tính toán bất kì khi nào nó được gọi, nên nếu biết tận dụng computed để tính toán các dữ liệu có sẵn thì sẽ cải thiện được performance app của các bạn. Điều tuyệt vời của **computed** là nó sẽ được cached nên giả sử bạn có 1 **computed** với hàng loạt tính toán, nhiều vòng lặp trong đó, mà nếu các biến phụ thuộc không thay đổi thì khi sử dụng nó sẽ chỉ mất thời gian tính 1 lần, những lần sau kết quả sẽ được sử dụng lại từ lần trước

2. Computed và Method

```
<div id="app">
  <button @click="a++">A++</button>
  <button @click="b++">B++</button>
  <p>Number + A={{addA}}</p>
  <p>Number + A={{addB}}</p>
</div>
```

```
computed:{
  addA:function()
  {
    console.log("addA")
    return this.a+this.number;

  },
  addB:function()
  {
    console.log("addB")
    return this.b+this.number;

  }
}
})
```

```
new Vue({
  el:'#app',
  data:{
    a:1,
    b:1,
    number:10
  },
  methods: {
    addA:function()
    {
      console.log("addA")
      return this.a+this.number;

    },
    addB:function()
    {
      console.log("addB")
      return this.b+this.number;

    }
  },
```

3. Computed với Getter và Setter

```vue
<script setup>
import { ref, computed } from 'vue'

const firstName = ref('John')
const lastName = ref('Doe')

const fullName = computed({
  // getter
  get() {
    return firstName.value + ' ' + lastName.value
  },
  // setter
  set(newValue) {
    // Note: we are using destructuring assignment syntax here.
    [firstName.value, lastName.value] = newValue.split(' ')
  }
})
</script>
```

4. Watcher trong Vue.js

- **Computed** có thể xử lý việc chỉ tính toán lại khi có sự thay đổi trên dữ liệu, giúp tiết kiệm được nhiều chi phí thì Vue cung cấp cho chúng ta một cách tổng quan hơn để xử lý những dữ liệu bị thay đổi đó là watcher
- **Watcher:** giúp chúng ta theo dõi sự thay đổi và sau đó thực hiện những tính toán phức tạp
- Cho một ví dụ như sau: khi click vào button thì giá trị của message sẽ thay đổi. Chúng ta them watcher vào message để mỗi lần giá trị của nó thay đổi thì sẽ in ra của sổ console 1 thông báo.

4. Watcher trong Vue.js

- **Trong Vue 3 có 2 cách để tạo**

1. Sử dụng watch API:

```
<script setup>
import { ref, watch } from 'vue';

const count = ref(0);

watch(count, (newValue, oldValue) => {
  console.log(`count đã thay đổi từ ${oldValue} thành ${newValue}`);
});
</script>
```

4. Watcher trong Vue.js

- 2. Sử dụng `watchEffect` API

```
<script setup>
import { ref, watchEffect } from 'vue';

const count = ref(0);

watchEffect(() => {
  console.log(`Giá trị hiện tại của count là: ${count.value}`);
});
</script>
```

4. Watcher trong Vue.js

```
<template>
    <div class="my-component">
        <div>{{ message }}</div>
        <div><button @click="changeMessage">Click me</button></div>
    </div>
</template>

<script>
    export default {
        data() {
            return {
                message: 'This is my first component using binding data'
            }
        },
        methods: {
            changeMessage() {
                this.message = 'this is new message'
            }
        },
        watch: {
            message() {
                console.log('message changed')
            }
        }
    }
</script>
```

4. Watcher trong Vue.js

- **Lưu ý:** tên của **watcher** phải trùng tên với **data**. Vậy watcher đúng như nghĩa của nó là theo dõi sự thay đổi của dữ liệu và thực thi hành động kèm theo
- **Computed** cũng sử dụng giống như những property trong data nên chúng ta cũng có thể watch computed
- Cũng ví dụ trên, chúng ta cùng xem watch computed như thế nào

4. Watcher trong Vue.js

```
<div id="demo">{{ fullName }}</div>
```

```
const vm = Vue.createApp({
  data() {
    return {
      firstName: 'Foo',
      lastName: 'Bar',
      fullName: 'Foo Bar'
    }
  },
  watch: {
    firstName(val) {
      this.fullName = val + ' ' + this.lastName
    },
    lastName(val) {
      this.fullName = this.firstName + ' ' + val
    }
  }
}).mount('#demo')
```

```
const vm = Vue.createApp({
  data() {
    return {
      firstName: 'Foo',
      lastName: 'Bar'
    }
  },
  computed: {
    fullName() {
      return this.firstName + ' ' + this.lastName
    }
  }
}).mount('#demo')
```

4. Watcher trong Vue.js

- **Sự khác biệt giữa watch và watchEffect**
  - **watch:**
  - ✓ Theo dõi một hoặc nhiều nguồn dữ liệu cụ thể.
  - ✓ Chỉ kích hoạt khi các nguồn dữ liệu được chỉ định thay đổi.
  - **watchEffect:**
  - ✓ Tự động theo dõi tất cả các nguồn dữ liệu được sử dụng bên trong callback.
  - ✓ Kích hoạt ngay lập tức khi callback được chạy lần đầu tiên.

4. Watcher trong Vue.js

- **Ví dụ watch:**

```
<script setup>
import { ref, watch } from 'vue';


const firstName = ref('John');
const lastName = ref('Doe');


watch([firstName, lastName], ([newFirstName, newLastName], [oldFirstName, oldLastName]) => {
  console.log(`Tên đã thay đổi từ ${oldFirstName} ${oldLastName} thành ${newFirstName} ${newLastName}`);
});
</script>
```

4. Watcher trong Vue.js

- **Ví dụ watchEffect:**

```
<script setup>
import { ref, watchEffect } from 'vue';

const firstName = ref('John');
const lastName = ref('Doe');

watchEffect(() => {
  console.log(`Tên đầy đủ là: ${firstName.value} ${lastName.value}`);
});
</script>
```

4. Watcher trong Vue.js

- **Trường hợp sử dụng của Watcher**

✓ Theo dõi biến đổi dữ liệu phức tạp: Khi bạn cần thực hiện một hành động mỗi khi một biến phức tạp thay đổi (như thực hiện gọi API).

✓ Quản lý side-effects: Thực hiện các tác vụ như lưu trữ dữ liệu, đồng bộ hóa trạng thái, hoặc kiểm soát UI.

4. Watcher trong Vue.js

- **Tính năng nâng cao của Watcher**

✓ Immediate Execution: Với watch, bạn có thể thiết lập watcher để kích hoạt ngay khi component được mount.

```
watch(count, callback, { immediate: true });
```

✓ Deep Watching: Theo dõi thay đổi sâu trong các object hoặc array

```
watch(complexObject, callback, { deep: true });
```

4. Watcher trong Vue.js

- **Lưu ý:**

✓ Không sử dụng watcher để tính toán giá trị từ các nguồn dữ liệu đơn giản. Thay vào đó, hãy sử dụng computed.

✓ Sử dụng watcher để quản lý side-effects hoặc thao tác không liên quan trực tiếp đến việc hiển thị trong giao diện.


the
end