계산된 속성

Vue School의 무료 동영상 강의 보기

기본 예제

템플릿 내 표현식은 매우 편리하지만 간단한 작업을 위한 것입니다. 템플릿에 너무 많은 논리를 넣으면 비대해져 유지 관리가 어려워질 수 있습니다. 예를 들어 객체 내 배열이 있는 경우:

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

그리고 `author` 가 이미 책을 가지고 있는지에 따라 다른 메시지를 표시하고 싶다면:

```template
<p>책을 가지고 있다:</p>
<span>{{ author.books.length > 0 ? 'Yes' : 'No' }}</span>
```

이 시점에서 템플릿이 약간 복잡해집니다. 우리가 여기서 인지해야 할 요점은 템플릿의 반응형 결과가 `author.books` 에 의해 계산된다는 점보다, 템플릿 내에서 이 코드를 두 번 이상 반복하고 싶지 않다는 것입니다.

따라서 반응형 데이터를 포함하는 복잡한 논리의 경우, **계산된 속성**을 사용하는 것이 좋습니다. 다음은 개선된 동일한 예제입니다:

```vue
<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'
  ]
})

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

<template>
  <p>책을 가지고 있다:</p>
  <span>{{ publishedBooksMessage }}</span>
</template>
```

온라인 연습장으로 실행하기

여기에서 우리는 `publishedBooksMessage` 라는 계산된 속성을 선언했습니다. `computed()` 함수는 **getter** 함수를 전달받기를 기대하며, 반환된 값은 **계산된 ref**입니다. 일반적인 ref와 유사하게, 계산된 결과에 `publishedBooksMessage.value` 로 접근할 수 있습니다. 계산된 ref는 템플릿에서 자동으로 풀리기 때문에 템플릿 표현식에서 `.value` 없이 참조할 수 있습니다.

계산된 속성은 의존된 반응형을 자동으로 추적합니다. Vue는 `publishedBooksMessage` 의 값이 `author.books` 에 의존한다는 것을 알고 있으므로, `author.books` 가 변경되면 `publishedBooksMessage` 를 바인딩해 의존하는 모든 것을 업데이트합니다.

참고: **computed** 타입 지정하기

계산된 캐싱 vs 메서드

표현식에서 메서드를 호출하여 동일한 결과를 얻을 수도 있습니다:

```template
<p>{{ calculateBooksMessage() }}</p>
```

```js
// 컴포넌트 내에서
function calculateBooksMessage() {
  return author.books.length > 0 ? 'Yes' : 'No'
}
```

계산된 속성 대신 메서드로 동일한 기능을 정의할 수 있습니다. 결과적으로 두 가지 접근 방식은 실제로 완전히 동일합니다. 그러나 차이점은 **계산된 속성은 의존된 반응형을 기반으로 캐시된다는 점**입니다. 계산된 속성은 의존된 반응형 중 일부가 변경된 경우에만 재평가됩니다. 즉, `author.books` 가 변경되지 않은 한 여러 곳에서 `publishedBooksMessage` 에 접근할 경우, getter 함수를 다시 실행하지 않고 이전에 계산된 결과를 즉시 반환합니다.

아래 예제는 `Date.now()` 가 반응형으로써 의존된 것이 아니기 때문에 이후 계산된 속성이 업데이트되지 않음을 의미합니다:

```js
const now = computed(() => Date.now())
```

이와 반대로 메서드 호출은 **리렌더링이 발생할 때마다 항상 함수를 실행**합니다.

캐싱이 필요한 이유는 무엇일까요? 거대한 배열을 루프 하며 많은 계산을 해야 하는 값비싼 비용의 `list` 속성이 있다고 가정해봅시다. 그리고 `list` 에 의존하는 또 다른 계산된 속성이 있을 수 있습니다. 캐싱이 없다면 우리는 `list` 의 getter를 불필요하게 많이 실행할 것입니다! 캐싱을 원하지 않는 경우에만 메서드 호출을 사용하십시오.

수정 가능한 계산된 속성

계산된 속성은 기본적으로 getter 전용입니다. 계산된 속성에 새 값을 할당하려고 하면 런타임 에러가 발생합니다. 드물게 "수정 가능한" 계산된 속성이 필요한 경우, getter와 setter를 모두 제공하여 속성을 만들 수 있습니다.

```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) {
```

```
    // 참고: 분해 할당 문법을 사용함.
    [firstName.value, lastName.value] = newValue.split(' ')
  }
})
</script>
```

이제 `fullName.value = 'John Doe'` 를 실행하면 setter가 호출되고, 그에 따라 `firstName` 과 `lastName` 이 업데이트됩니다.

모범 사례

getter에서 사이드 이펙트는 금물

계산된 getter 함수는 순수한 계산만을 수행하고 부작용이 없어야 한다는 것을 기억하는 것이 중요합니다. 예를 들어, **계산된 getter 안에서 다른 상태를 변형시키거나, 비동기 요청을 하거나, DOM을 변경하는 행위는 하지 마세요!** 계산된 속성은 다른 값들을 기반으로 값을 파생시키는 방법을 선언적으로 설명하는 것으로 생각해야 합니다 - 그것의 유일한 책임은 그 값을 계산하고 반환하는 것입니다. 가이드에서 나중에 우리는 상태 변화에 대한 반응으로 부작용을 수행하는 방법에 대해 **watchers**를 사용하여 논의할 것입니다.

계산된 값을 변경하지 마십시오

계산된 속성에서 반환된 값은 파생된 상태입니다. 임시 스냅샷으로 생각하십시오. 소스 상태가 변경될 때마다 새 스냅샷이 생성됩니다. 스냅샷을 변경하는 것은 의미가 없으므로 계산된 반환 값은 읽기 전용으로 처리되어야 하며 변경되지 않아야 합니다. 대신 새 계산을 트리거하기 위해 의존하는 소스 상태를 업데이트하십시오.