계산된 속성

Vue School의 무료 동영상 강의 보기

기본 예제

템플릿 내 표현식은 매우 편리하지만, 단순한 연산을 위한 것입니다. 템플릿에 너무 많은 로직을 넣으면 템플릿이 복잡해지고 유지보수가 어려워질 수 있습니다. 예를 들어, 중첩 배열이 있는 객체가 있다고 가정해봅시다:

```js
const author = reactive({
  name: 'John Doe',
  books: [
    'Vue 2 - 고급 가이드',
    'Vue 3 - 기본 가이드',
    'Vue 4 - 미스터리'
  ]
})
```

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

```template
<p>출판한 책이 있습니까?</p>
<span>{{ author.books.length > 0 ? '예' : '아니오' }}</span>
```

이 시점에서 템플릿이 약간 복잡해지고 있습니다. `author.books` 에 따라 계산이 수행된다는 것을 알아차리기 위해 잠시 들여다봐야 합니다. 더 중요한 것은, 이 계산을 템플릿에서 여러 번 포함해야 한다면 반복하지 않기를 원할 것입니다.

이런 이유로, 반응형 데이터를 포함하는 복잡한 로직에는 **계산된 속성**을 사용하는 것이 권장됩니다. 다음은 동일한 예제를 리팩토링한 것입니다:

```
import { reactive, computed } from 'vue'

const author = reactive({
  name: 'John Doe',
  books: [
    'Vue 2 - 고급 가이드',
    'Vue 3 - 기본 가이드',
    'Vue 4 - 미스터리'
  ]
})

// 계산된 ref
const publishedBooksMessage = computed(() => {
  return author.books.length > 0 ? '예' : '아니오'
})
</script>

<template>
  <p>출판한 책이 있습니까?</p>
  <span>{{ publishedBooksMessage }}</span>
</template>
```

Playground에서 실행해보기

여기서 우리는 계산된 속성 `publishedBooksMessage` 를 선언했습니다. `computed()` 함수는 getter 함수를 인자로 받으며, 반환값은 **계산된 ref**입니다. 일반 ref와 유사하게, 계산된 결과는 `publishedBooksMessage.value` 로 접근할 수 있습니다. 계산된 ref는 템플릿에서 자동으로 언래핑되므로, 템플릿 표현식에서는 `.value` 없이 참조할 수 있습니다.

계산된 속성은 자동으로 자신의 반응형 의존성을 추적합니다. Vue는 `publishedBooksMessage` 의 계산이 `author.books` 에 의존한다는 것을 알고 있으므로, `author.books` 가 변경될 때 `publishedBooksMessage` 에 의존하는 모든 바인딩을 업데이트합니다.

참고: 계산된 속성 타입 지정 TS

계산된 속성 캐싱 vs. 메서드

표현식에서 메서드를 호출하여 동일한 결과를 얻을 수 있다는 것을 눈치챘을 수도 있습니다:

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

template

```
// 컴포넌트 내에서
function calculateBooksMessage() {
```

js

```
}
```

계산된 속성 대신 동일한 함수를 메서드로 정의할 수 있습니다. 최종 결과는 두 접근 방식이 실제로 완전히 동일합니다. 그러나 **계산된 속성은 자신의 반응형 의존성에 따라 캐시됩니다.** 계산된 속성은 반응형 의존성 중 일부가 변경될 때만 다시 평가됩니다. 즉, `author.books` 가 변경되지 않는 한, `publishedBooksMessage` 에 여러 번 접근해도 getter 함수를 다시 실행하지 않고 이전에 계산된 결과를 즉시 반환합니다.

이것은 또한 다음과 같은 계산된 속성은 절대 업데이트되지 않는다는 것을 의미합니다. 왜냐하면 `Date.now()` 는 반응형 의존성이 아니기 때문입니다:

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

반면, 메서드 호출은 리렌더가 발생할 때마다 **항상** 함수를 실행합니다.

왜 캐싱이 필요할까요? 예를 들어, 대용량 배열을 반복하고 많은 계산을 수행해야 하는 비용이 큰 계산된 속성 `list` 가 있다고 가정해봅시다. 그리고 다른 계산된 속성이 다시 `list` 에 의존할 수 있습니다. 캐싱이 없다면, `list` 의 getter를 불필요하게 여러 번 실행하게 됩니다! 캐싱이 필요하지 않은 경우에는 메서드 호출을 대신 사용하세요.

쓰기 가능한 계산된 속성

계산된 속성은 기본적으로 getter 전용입니다. 계산된 속성에 새 값을 할당하려고 하면 런타임 경고가 발생합니다. "쓰기 가능한" 계산된 속성이 필요한 드문 경우에는 getter와 setter를 모두 제공하여 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: 여기서는 구조 분해 할당 문법을 사용하고 있습니다.
    [firstName.value, lastName.value] = newValue.split(' ')
  }
```

```
</script>
```

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

이전 값 가져오기

3.4+에서만 지원

필요한 경우, 계산된 속성 getter의 첫 번째 인자를 통해 계산된 속성이 반환한 이전 값을 가져올 수 있습니다:

vue

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

const count = ref(2)

// 이 계산된 속성은 count가 3 이하일 때 count 값을 반환합니다.
// count가 4 이상이 되면, 조건을 만족했던 마지막 값을 대신 반환합니다.
// count가 다시 3 이하가 될 때까지 이전 값을 반환합니다.
const alwaysSmall = computed((previous) => {
  if (count.value <= 3) {
    return count.value
  }

  return previous
})
</script>
```

쓰기 가능한 계산된 속성을 사용하는 경우:

vue

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

const count = ref(2)

const alwaysSmall = computed({
  get(previous) {
    if (count.value <= 3) {
      return count.value
    }

    return previous
  },
  set(newValue) {
```

```
  }
})
</script>
```

모범 사례

Getter는 부작용이 없어야 합니다

계산된 getter 함수는 순수 계산만 수행하고 부작용이 없어야 한다는 점을 기억하는 것이 중요합니다. 예를 들어, **다른 상태를 변경하거나, 비동기 요청을 하거나, 계산된 getter 내부에서 DOM을 변경하지 마세요!** 계산된 속성은 다른 값을 기반으로 값을 도출하는 방법을 선언적으로 설명하는 것으로 생각하세요. 그 유일한 책임은 해당 값을 계산하고 반환하는 것입니다. 가이드의 뒷부분에서 watchers를 사용하여 상태 변경에 반응하여 부작용을 수행하는 방법에 대해 다룰 것입니다.

계산된 값 변경 피하기

계산된 속성에서 반환된 값은 파생 상태입니다. 이를 임시 스냅샷으로 생각하세요. 소스 상태가 변경될 때마다 새로운 스냅샷이 생성됩니다. 스냅샷을 변경하는 것은 의미가 없으므로, 계산된 반환 값은 읽기 전용으로 취급하고 절대 변경하지 마세요. 대신, 새로운 계산을 트리거하려면 해당 값이 의존하는 소스 상태를 업데이트하세요.

Previous
반응성 기초

Next
클래스 및 스타일 바인딩