트랜지션

Vue는 상태 변화에 대응하기 위해 트랜지션 및 애니메이션 작업에 도움이 되는 두 가지 빌트인 컴포넌트를 제공합니다:

엘리먼트 또는 컴포넌트가 DOM에 들어오고 나갈 때 애니메이션을 적용하기 위한 <Transition> . 이 페이지에서 다룹니다.

엘리먼트 또는 컴포넌트가 v-for 리스트에 삽입, 제거 또는 이동할 때 애니메이션을 적용하기 위한 <TransitionGroup> . 이것은 다음 장에서 다룹니다.

이 두 가지 컴포넌트 외에도 CSS 클래스 트랜지션 또는 스타일 바인딩을 통한 상태 기반 애니메이션과 같은 기술을 사용하여 Vue에서 애니메이션을 적용할 수도 있습니다. 이러한 추가 기술은 애니메이션 기법 장에서 다룹니다.

<Transition> 컴포넌트

<Transition> 은 빌트인 컴포넌트이므로 등록하지 않고도 컴포넌트의 템플릿에서 사용할 수 있습니다. 기본 슬롯을 통해 전달된 엘리먼트 또는 컴포넌트에 진입(enter) 및 진출(leave) 애니메이션을 적용하는 데 사용할 수 있습니다. 해당 애니메이션은 다음 중 하나의 조건에 충족하면 발생합니다:

v-if 를 통한 조건부 렌더링
v-show 를 통한 조건부 표시
스페셜 엘리먼트 <component> 를 통해 전환되는 동적 컴포넌트
key 라는 특수한 속성 값 변경

다음은 가장 기본적인 사용법 예제입니다:

template

```
<button @click="show = !show">토글</button>
<Transition>
  <p v-if="show">안녕</p>
</Transition>
```

css

```
/* 이 클래스들이 무엇을 하는지 다음에 설명하겠습니다. */
.v-enter-active,
.v-leave-active {
  transition: opacity 0.5s ease;
}

.v-enter-from,
.v-leave-to {
  opacity: 0;
}
```

토글

안녕

온라인 연습장으로 실행하기

TIP

<Transition> 은 슬롯 컨텐츠로 단일 엘리먼트 또는 컴포넌트만 지원합니다. 컨텐츠가 컴포넌트인 경우, 컴포넌트에는 단일 루트 엘리먼트만 있어야 합니다.

<Transition> 컴포넌트 안에 엘리먼트가 삽입되거나 제거되면 다음과 같은 일이 일어납니다:

Vue는 대상 엘리먼트에 CSS 트랜지션 또는 애니메이션이 적용되었는지 여부를 자동으로 감지합니다. 그리고 적절한 타이밍에 여러 CSS 트랜지션 클래스가 추가/제거됩니다.

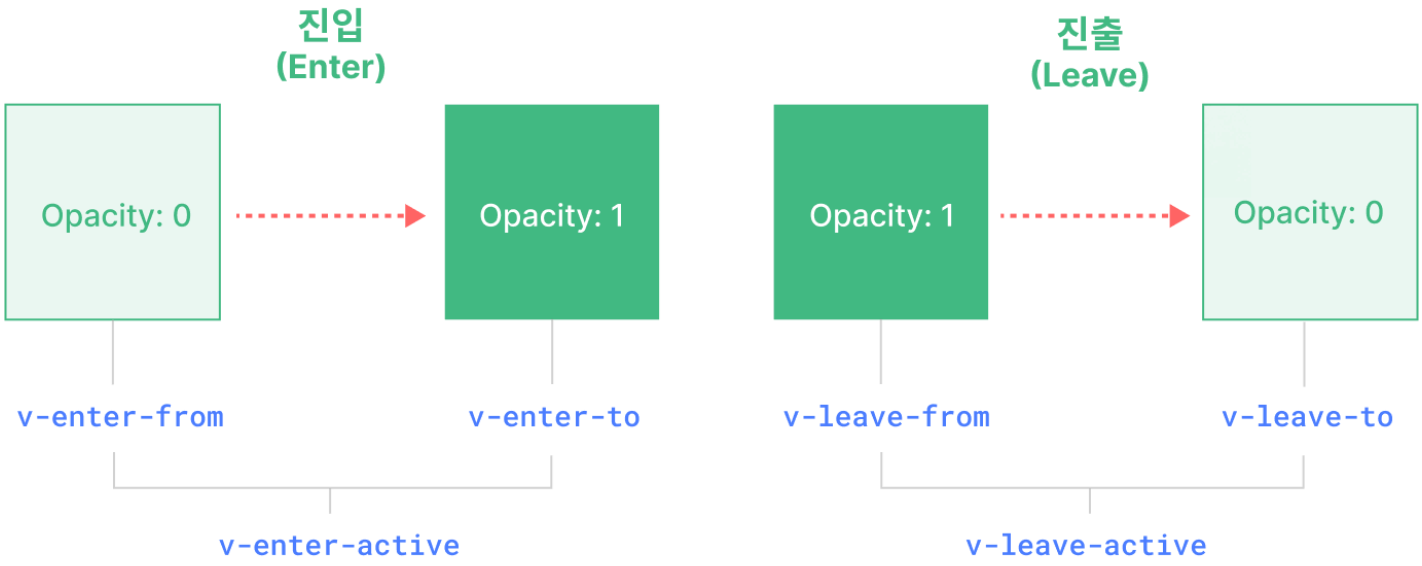
JavaScript 훅에 대한 리스너가 있는 경우, 이 훅은 적절한 타이밍에 호출됩니다.

CSS 트랜지션/애니메이션이 감지되지 않고 JavaScript 훅이 제공되지 않으면, DOM 삽입/제거 작업이 브라우저의 다음 애니메이션 프레임에서 실행됩니다.

CSS 기반 트랜지션

트랜지션 클래스

진입/진출 트랜지션에 적용되는 6개의 클래스가 있습니다.



v-enter-from : 진입 시작 상태. 엘리먼트가 삽입되기 전에 추가되고, 엘리먼트가 삽입되고 1 프레임 후 제거됩니다.

v-enter-active : 진입 활성 상태. 모든 진입 상태에 적용됩니다. 엘리먼트가 삽입되기 전에 추가되고, 트랜지션/애니메이션이 완료되면 제거됩니다. 이 클래스는 진입 트랜지션에 대한 지속 시간, 딜레이 및 이징(easing) 곡선을 정의하는 데 사용할 수 있습니다.

v-enter-to : 진입 종료 상태. 엘리먼트가 삽입된 후 1 프레임 후 추가되고(동시에 v-enter-from 이 제거됨), 트랜지션/애니메이션이 완료되면 제거됩니다.

v-leave-from : 진출 시작 상태. 진출 트랜지션이 트리거되면 즉시 추가되고 1 프레임 후 제거됩니다.

v-leave-active : 진출 활성 상태. 모든 진출 상태에 적용됩니다. 진출 트랜지션이 트리거되면 즉시 추가되고, 트랜지션/애니메이션이 완료되면 제거됩니다. 이 클래스는 진출 트랜지션에 대한 지속 시간, 딜레이 및 이징 곡선을 정의하는 데 사용할 수 있습니다.

v-leave-to : 진출 종료 상태. 진출 트랜지션이 트리거된 후 1 프레임이 추가되고(동시에 v-leave-from 이 제거됨), 트랜지션/애니메이션이 완료되면 제거됩니다.

v-enter-active 및 v-leave-active 는 진입/진출 트랜지션에 대해 다른 이징 곡선을 지정할 수 있는 기능을 제공합니다. 이에 대한 예는 다음 섹션에서 볼 수 있습니다.

트랜지션 이름 지정하기

트랜지션은 name prop을 통해 이름을 지정할 수 있습니다:

template

```
<Transition name="fade">
  ...
</Transition>
```

이름이 지정된 트랜지션의 경우, 트랜지션 클래스에는 `v` 대신 이름이 접두사로 붙습니다. 예를 들어 위의 트랜지션에 적용된 클래스는 `v-enter-active` 대신 `fade-enter-active` 가 됩니다. 페이드 트랜지션을 위한 CSS는 다음과 같아야 합니다:

css

```
.fade-enter-active,
.fade-leave-active {
  transition: opacity 0.5s ease;
}

.fade-enter-from,
.fade-leave-to {
  opacity: 0;
}
```

CSS 트랜지션

위의 기본 예제에서 볼 수 있듯이 일반적으로 `<Transition>` 은 네이티브 **CSS** 트랜지션과 함께 사용됩니다. `transition` CSS 속성은 애니메이션을 적용해야 하는 속성, 트랜지션 기간 및 이징 곡선을 포함하여 트랜지션 관련된 여러 속성을 지정할 수 있게 해주는 약칭입니다.

다음은 다양한 진입/진출 트랜지션/애니메이션을 구현하기 위해 지속 시간 및 이징 곡선을 사용하는 고급 예제입니다:

template

```
<Transition name="slide-fade">
  <p v-if="show">안녕</p>
</Transition>
```

css

```
/*
  진입/진출 애니메이션은 다른 지속 시간과
  타이밍 함수를 사용할 수 있습니다.
*/
.slide-fade-enter-active {
  transition: all 0.3s ease-out;
}

.slide-fade-leave-active {
  transition: all 0.8s cubic-bezier(1, 0.5, 0.8, 1);
}

.slide-fade-enter-from,
.slide-fade-leave-to {
  transform: translateX(20px);
  opacity: 0;
}
```

토글: 슬라이드 + 페이드

안녕

온라인 연습장으로 실행하기

CSS 애니메이션

네이티브 **CSS** 애니메이션은 CSS 트랜지션과 동일한 방식으로 적용되지만, 엘리먼트가 삽입된 직후에 `*-enter-from` 이 제거되지 않고, `animationend` 이벤트에서 제거된다는 차이점이 있습니다.

대부분의 CSS 애니메이션의 경우 `*-enter-active` 및 `*-leave-active` 클래스에서 간단히 선언할 수 있습니다. 다음은 예제입니다:

template

```
<Transition name="bounce">
  <p v-if="show" style="text-align: center;">
    안녕! 여기에 탄력적인 텍스트가 있어요!
  </p>
</Transition>
```

css

```
.bounce-enter-active {
  animation: bounce-in 0.5s;
}
.bounce-leave-active {
  animation: bounce-in 0.5s reverse;
}
@keyframes bounce-in {
  0% {
    transform: scale(0);
  }
  50% {
    transform: scale(1.25);
  }
  100% {
    transform: scale(1);
  }
}
```

Toggle

Hello here is some bouncy text!

온라인 연습장으로 실행하기

커스텀 트랜지션 클래스

아래 나열된 `<Transition>` props에 커스텀 트랜지션 클래스를 정의하여 전달할 수도 있습니다:

- enter-from-class
- enter-active-class
- enter-to-class
- leave-from-class
- leave-active-class
- leave-to-class

이들은 기존의 클래스 이름을 재정의합니다. 이는 Vue의 트랜지션 시스템을 **Animate.css**와 같은 기존 CSS 애니메이션 라이브러리와 결합하려는 경우에 특히 유용합니다.

template

```
<!-- Animate.css가 페이지에 포함되어 있다고 가정합니다. -->
<Transition
  name="custom-classes"
  enter-active-class="animate__animated animate__tada"
  leave-active-class="animate__animated animate__bounceOutRight"
>
  <p v-if="show">안녕</p>
</Transition>
```

온라인 연습장으로 실행하기

트랜지션과 애니메이션을 같이 사용하기

Vue는 트랜지션이 종료된 시점을 알기 위해 이벤트 리스너를 연결해야 합니다. 리스너는 적용된 CSS 규칙 유형에 따라 `transitionend` 또는 `animationend` 가 될 수 있습니다. 둘 중 하나만 사용하는 경우, Vue는 올바른 유형을 자동으로 감지할 수 있습니다.

그러나 경우에 따라 하나의 엘리먼트에 Vue에 의해 트리거된 CSS 애니메이션과 hover에 대한 CSS 트랜지션 효과가 같이 있을 수 있습니다. 이러한 경우에는 `type` prop을 사용하여 `animation` 또는 `transition` 값을 전달하여 Vue에서 처리할 유형을 명시적으로 선언해야 합니다:

template
```
<Transition type="animation">...</Transition>
```

중첩된 트랜지션과 지속시간 설정하기

트랜지션 클래스는 `<Transition>` 의 직접적인 자식 엘리먼트에만 적용되지만, 중첩된 CSS 선택기를 사용하여 중첩된 엘리먼트를 트랜지션할 수 있습니다:

template
```
<Transition name="nested">
  <div v-if="show" class="outer">
    <div class="inner">
      안녕
    </div>
  </div>
</Transition>
```

css
```
/* 중첩 엘리먼트를 대상으로 하는 규칙 */
.nested-enter-active .inner,
.nested-leave-active .inner {
  transition: all 0.3s ease-in-out;
}

.nested-enter-from .inner,
.nested-leave-to .inner {
  transform: translateX(30px);
  opacity: 0;
}

/* ... 다른 필요한 CSS들은 생략합니다... */
```

진입 시 중첩된 엘리먼트에 트랜지션 딜레이를 추가하여, 시차가 있는 진입 애니메이션 순서를 생성할 수도 있습니다:

css
```
/* 시차를 둔 중첩 엘리먼트의 딜레이된 진입 효과 */
.nested-enter-active .inner {
  transition-delay: 0.25s;
}
```

그러나 이것은 작은 문제를 만듭니다. 기본적으로 `<Transition>` 컴포넌트는 루트 트랜지션 엘리먼트에서 첫 번째 `transitionend` 또는 `animationend` 이벤트를 수신하여 트랜지션이 완료된 시점을 자동으로 파악하려고 시도합니다. 중첩 트랜지션을 사용하면 모든 내부 엘리먼트의 트랜지션이 완료될 때까지 원하는 동작이 대기해야 합니다.

이러한 경우 `<transition>` 컴포넌트의 `duration` prop을 사용하여 명시적 트랜지션 지속 시간(밀리초 단위)을 지정할 수 있습니다. 총 지속 시간은 딜레이에 내부 엘리먼트의 트랜지션 지속 시간을 더한 값과 일치해야 합니다:

template
```
<Transition :duration="550">...</Transition>
```

토글

안녕

온라인 연습장으로 실행하기

필요한 경우 객체를 사용하여 진입/출 지속 시간에 대해 별도의 값을 지정할 수도 있습니다:

template

```
<Transition :duration="{ enter: 500, leave: 800 }">...</Transition>
```

성능 고려사항

위에 표시된 애니메이션은 대부분 `transform` 및 `opacity` 와 같은 속성을 사용하고 있음을 알 수 있습니다. 이러한 속성은 다음과 같은 이유로 애니메이션에 효율적입니다:

애니메이션 진행중에 문서 레이아웃에 영향을 미치지 않으므로 모든 애니메이션 프레임에서 값비싼 CSS 레이아웃 계산을 트리거하지 않습니다.

대부분의 최신 브라우저는 `transform` 에 애니메이션을 적용할 때 GPU 하드웨어 가속을 활용할 수 있습니다.

이에 비해 `height` 나 `margin` 같은 속성은 CSS 레이아웃을 트리거하므로 애니메이션하기에 훨씬 비용이 많이 들며, 신중하게 사용해야 합니다.

JavaScript 훅

`<Transition>` 컴포넌트의 이벤트를 수신하여 JavaScript로 트랜지션 프로세스에 연결할 수 있습니다:

html

```
<Transition
  @before-enter="onBeforeEnter"
  @enter="onEnter"
  @after-enter="onAfterEnter"
  @enter-cancelled="onEnterCancelled"
  @before-leave="onBeforeLeave"
  @leave="onLeave"
  @after-leave="onAfterLeave"
  @leave-cancelled="onLeaveCancelled"
>
  <!-- ... -->
</Transition>
```

js

```
// 엘리먼트가 DOM에 삽입되기 전에 호출됩니다.
// 이것을 사용하여 엘리먼트의 "enter-from" 상태를 설정합니다.
function onBeforeEnter(el) {}

// 엘리먼트가 삽입되고 1 프레임 후 호출됩니다.
// 진입 애니메이션을 시작하는 데 사용합니다.
function onEnter(el, done) {
  // CSS와 함께 사용되는 경우, 선택적으로
  // 트랜지션 종료를 나타내기 위해 done 콜백을 호출합니다.
  done()
}

// 진입 트랜지션이 완료되면 호출됩니다.
function onAfterEnter(el) {}

// 진입 트랜지션 취소가 완료되기 전 호출됩니다.
function onEnterCancelled(el) {}
```

```
// 진출 훅 전에 호출됩니다.
// 대부분의 경우 그냥 진출 훅을 사용해야 합니다.
function onBeforeLeave(el) {}

// 진출 트랜지션이 시작될 때 호출됩니다.
// 진출 애니메이션을 시작하는 데 사용합니다.
function onLeave(el, done) {
  // CSS와 함께 사용되는 경우, 선택적으로
  // 트랜지션 종료를 나타내기 위해 done 콜백을 호출합니다.
  done()
}

// 진출 트랜지션이 완료되고,
// 엘리먼트가 DOM에서 제거된 후 호출됩니다.
function onAfterLeave(el) {}

// v-show 트랜지션에서만 사용 가능합니다.
function onLeaveCancelled(el) {}
```

이 훅은 CSS 트랜지션/애니메이션과 함께 사용하거나 단독으로 사용할 수 있습니다.

JavaScript 전용 트랜지션을 사용할 때 일반적으로 `:css="false"` prop을 추가하는 것이 좋습니다. 이것은 Vue가 CSS 트랜지션을 자동으로 감지하는 것을 건너뛰도록 명시적으로 지시합니다. 성능이 약간 향상되는 것 외에도 CSS 규칙이 실수로 트랜지션을 방해하는 것을 방지합니다.

template

```
<Transition
  ...
  :css="false"
>
  ...
</Transition>
```

`:css="false"` 를 사용하면 트랜지션이 끝나는 시점 제어에 대한 전적인 책임을 가집니다. 이 경우 `@enter` 및 `@leave` 훅에 `done` 콜백이 필요합니다. 그렇지 않으면 훅이 동기적으로 호출되고 트랜지션이 즉시 완료됩니다.

다음은 **GSAP** 라이브러리를 사용하여 애니메이션을 수행하는 데모입니다. 물론 **Anime.js**나 **Motion One**과 같은 다른 애니메이션 라이브러리를 사용할 수도 있습니다:

토글

온라인 연습장으로 실행하기

재사용 가능한 트랜지션

Vue의 컴포넌트 시스템을 통해 트랜지션을 재사용할 수 있습니다. 재사용 가능한 트랜지션을 만들기 위해 `<Transition>` 컴포넌트를 래핑하고 슬롯 컨텐츠를 전달하는 컴포넌트를 만들 수 있습니다:

vue

```
<!-- MyTransition.vue -->
<script>
// JavaScript 훅 로직...
</script>

<template>
  <!-- 빌트인 트랜지션 컴포넌트 래핑 -->
  <Transition
    name="my-transition"
```

```
    @enter="onEnter"
    @leave="onLeave">
    <slot></slot> <!-- 슬롯 컨텐츠 전달 -->
  </Transition>
</template>

<style>
/*
  필요한 CSS 정의...
  참고: 슬롯 컨텐츠에 적용되지 않으므로,
  여기에서 <style scoped>를 사용하지 마십시오.
*/
</style>
```

이제 빌트인 버전처럼 MyTransition 을 가져와서 사용할 수 있습니다:

template

```
<MyTransition>
  <div v-if="show">안녕</div>
</MyTransition>
```

등장 트랜지션

노드의 초기 렌더링에도 트랜지션을 적용하려면 appear prop을 추가할 수 있습니다:

template

```
<Transition appear>
  ...
</Transition>
```

엘리먼트 간 트랜지션

v-if / v-show 로 앨리먼트를 토글하는 것 외에도, 특정 순간에 표시되는 앨리먼트가 하나만 있는지 확인하는 한 v-if / v-else / v-else-if 를 사용하여 두 앨리먼트 사이를 전환할 수도 있습니다:

template

```
<Transition>
  <button v-if="docState === 'saved'">수정</button>
  <button v-else-if="docState === 'edited'">저장</button>
  <button v-else-if="docState === 'editing'">취소</button>
</Transition>
```

클릭 시 상태 변경: **수정**

온라인 연습장으로 실행하기

트랜지션 모드

이전 예제에서 진입 시 엘리먼트와 진출 시 엘리먼트는 동시에 애니메이션되며 두 엘리먼트가 DOM에 있을 때 레이아웃 문제를 피하기 위해 position: absolute 로 만들어야 했습니다.

그러나 어떤 경우에는 이것이 선택가능한 옵션이 아니거나, 원하는 동작이 아닐수 있습니다. 진출 시 엘리먼트가 먼저 애니메이션 처리되고 진입 시 엘리먼트가 **진출 애니메이션이 완료된 이후에 삽입**되기를 원할 수 있습니다. 이러한 애니메이션을 수동으로 조정하는 것은 매우 복잡합니다. 운 좋게도 <Transition> 에 mode prop을 전달하여 이 동작을 활성화할 수 있습니다.

template

```
<Transition mode="out-in">
  ...
</Transition>
```

다음은 mode="out-in" 을 사용한 이전 데모입니다:

클릭 시 상태 변경: **수정**

<Transition> 은 자주 사용되지는 않지만 mode="in-out" 도 지원합니다.

컴포넌트 간 트랜지션

<Transition> 은 동적 컴포넌트에서도 사용할 수 있습니다:

template

```
<Transition name="fade" mode="out-in">
  <component :is="activeComponent"></component>
</Transition>
```

A B
A 컴포넌트

온라인 연습장으로 실행하기

동적 트랜지션

<Transition> name 과 같은 prop도 동적일 수 있습니다! 이를 통해 상태 변경에 따라 다른 트랜지션을 동적으로 적용할 수 있습니다:

template

```
<Transition :name="transitionName">
  <!-- ... -->
</Transition>
```

이것은 Vue의 트랜지션 클래스 규칙을 사용하여 CSS 트랜지션/애니메이션을 정의하고 둘 사이를 트랜지션하려는 경우에 유용할 수 있습니다.

컴포넌트의 현재 상태를 기반으로 JavaScript 트랜지션 훅에서 다른 동작을 적용할 수도 있습니다. 마지막으로, 동적 트랜지션을 만드는 궁극적인 방법은 사용할 트랜지션의 특성을 변경하기 위해 prop을 허용하는 재사용 가능한 트랜지션 컴포넌트를 사용하는 것입니다. 진부하게 들릴지 모르지만, 실제로 한계는 당신의 상상력뿐입니다.

key 속성을 사용한 트렌지션

때로는 트렌지션을 발생시키기 위해 DOM 엘리먼트를 강제로 다시 렌더링해야 할 필요가 있습니다.

예를 들어, 이 카운터 컴포넌트를 봅시다:

vue

```vue
<script setup>
import { ref } from 'vue';
const count = ref(0);

setInterval(() => count.value++, 1000);
</script>

<template>
  <Transition>
    <span :key="count">{{ count }}</span>
  </Transition>
</template>
```

key 속성(attribute)을 생략했다면, 텍스트 노드만 업데이트되고 트렌지션이 발생하지 않았을 것입니다. 그러나 key 속성을 사용함으로써, Vue는 count 가 변경될 때마다 새로운 span 엘리먼트를 생성하고 Transition 컴포넌트는 두 개의 다른 엘리먼트 사이에서 트렌지션을 할 수 있습니다.

온라인 연습장으로 실행하기

관련 문서

- **<Transition> API** 참고