



# 트랜지션

Vue는 상태 변화에 따라 트랜지션과 애니메이션을 다루는 데 도움이 되는 두 개의 내장 컴포넌트를 제공합니다:

`<Transition>` : 요소나 컴포넌트가 DOM에 진입하거나 퇴장할 때 애니메이션을 적용합니다. 이 페이지에서 다룹니다.

`<TransitionGroup>` : `v-for` 리스트에서 요소나 컴포넌트가 삽입, 제거, 이동될 때 애니메이션을 적용합니다. 다음 장에서 다룹니다.

이 두 컴포넌트 외에도, CSS 클래스 토글이나 스타일 바인딩을 통한 상태 기반 애니메이션 등 다른 기법을 사용하여 Vue에서 애니메이션을 적용할 수 있습니다. 이러한 추가 기법들은 [애니메이션 기법](#) 장에서 다룹니다.

## <Transition> 컴포넌트

`<Transition>` 은 내장 컴포넌트입니다: 즉, 어떤 컴포넌트의 템플릿에서도 별도의 등록 없이 사용할 수 있습니다. 기본 슬롯을 통해 전달된 요소나 컴포넌트에 진입 및 퇴장 애니메이션을 적용할 수 있습니다. 진입 또는 퇴장은 다음 중 하나에 의해 트리거될 수 있습니다:

`v-if` 를 통한 조건부 렌더링

`v-show` 를 통한 조건부 표시

`<component>` 특수 요소를 통한 동적 컴포넌트 토글

특수 `key` 속성 변경

가장 기본적인 사용 예시는 다음과 같습니다:

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

template



```
.v-enter-active,  
.v-leave-active {  
  transition: opacity 0.5s ease;  
}  
  
.v-enter-from,  
.v-leave-to {  
  opacity: 0;  
}
```

Toggle Fade

hello

### ▶ 플레이그라운드에서 실행해보기

#### ① TIP

<Transition> 은 슬롯 콘텐츠로 단일 요소 또는 컴포넌트만 지원합니다. 콘텐츠가 컴포넌트인 경우, 해당 컴포넌트 역시 단일 루트 요소만 가져야 합니다.

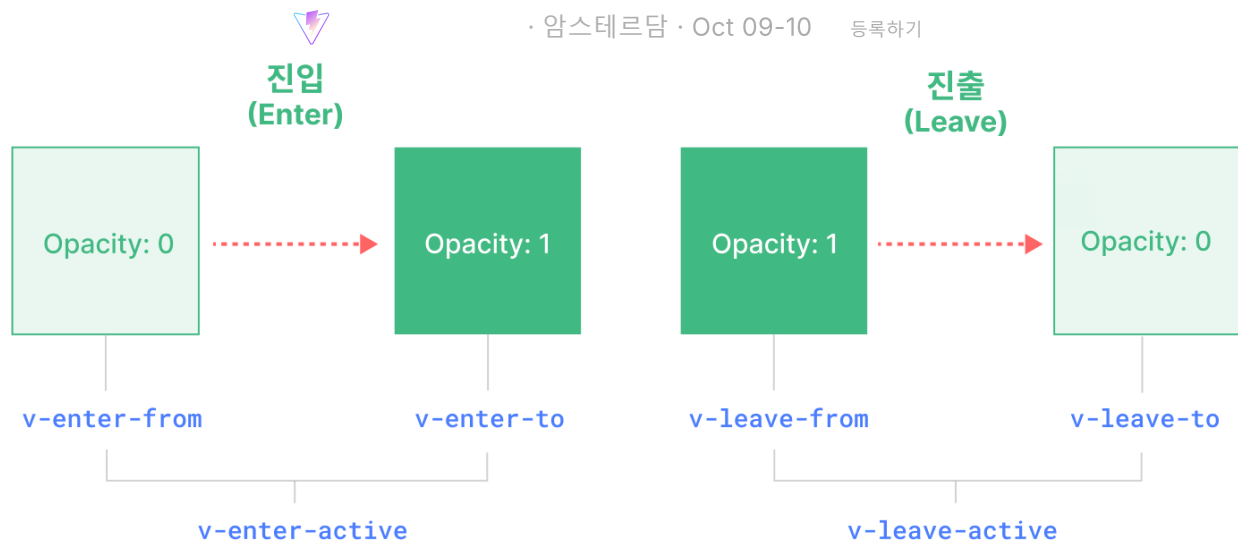
<Transition> 컴포넌트 내의 요소가 삽입되거나 제거될 때 다음과 같은 일이 발생합니다:

1. Vue는 대상 요소에 CSS 트랜지션 또는 애니메이션이 적용되어 있는지 자동으로 감지합니다. 적용되어 있다면, **CSS 트랜지션 클래스**들이 적절한 타이밍에 추가/제거됩니다.
2. 자바스크립트 훅에 대한 리스너가 있다면, 이 훅들이 적절한 타이밍에 호출됩니다.
3. CSS 트랜지션/애니메이션이 감지되지 않고 자바스크립트 훅도 제공되지 않은 경우, 삽입 및/또는 제거에 대한 DOM 조작이 브라우저의 다음 애니메이션 프레임에 실행됩니다.

## CSS 기반 트랜지션

### 트랜지션 클래스

진입/퇴장 트랜지션에는 여섯 개의 클래스가 적용됩니다.



1. `v-enter-from` : 진입의 시작 상태. 요소가 삽입되기 전에 추가되고, 요소가 삽입된 한 프레임 후에 제거됩니다.
2. `v-enter-active` : 진입의 활성 상태. 전체 진입 단계 동안 적용됩니다. 요소가 삽입되기 전에 추가되고, 트랜지션/애니메이션이 끝나면 제거됩니다. 이 클래스는 진입 트랜지션의 지속 시간, 지연, 이징 곡선을 정의하는 데 사용할 수 있습니다.
3. `v-enter-to` : 진입의 종료 상태. 요소가 삽입된 한 프레임 후( `v-enter-from` 이 제거되는 시점) 추가되고, 트랜지션/애니메이션이 끝나면 제거됩니다.
4. `v-leave-from` : 퇴장의 시작 상태. 퇴장 트랜지션이 트리거되자마자 추가되고, 한 프레임 후에 제거됩니다.
5. `v-leave-active` : 퇴장의 활성 상태. 전체 퇴장 단계 동안 적용됩니다. 퇴장 트랜지션이 트리거되자마자 추가되고, 트랜지션/애니메이션이 끝나면 제거됩니다. 이 클래스는 퇴장 트랜지션의 지속 시간, 지연, 이징 곡선을 정의하는 데 사용할 수 있습니다.
6. `v-leave-to` : 퇴장의 종료 상태. 퇴장 트랜지션이 트리거된 한 프레임 후( `v-leave-from` 이 제거되는 시점) 추가되고, 트랜지션/애니메이션이 끝나면 제거됩니다.

`v-enter-active` 와 `v-leave-active` 를 사용하면 진입/퇴장 트랜지션에 서로 다른 이징 곡선을 지정할 수 있습니다. 다음 섹션에서 예시를 확인할 수 있습니다.

## 네임드 트랜지션

트랜지션은 `name` prop을 통해 이름을 지정할 수 있습니다:

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

template



어, 위 트랜지션에 적용되는 클래스는 `v-enter-active` 대신 `fade-enter-active` 가 됩니다. 페이드 트랜지션의 CSS는 다음과 같습니다:

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

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

CSS

## CSS 트랜지션

<Transition> 은 네이티브 CSS 트랜지션과 함께 가장 자주 사용됩니다. 위의 기본 예시에서 볼 수 있습니다. `transition` CSS 속성은 트랜지션의 여러 측면(애니메이션할 속성, 트랜지션 지속 시간, 이징 곡선 등)을 지정할 수 있는 단축 속성입니다.

다음은 여러 속성을 트랜지션하고, 진입과 퇴장에 서로 다른 지속 시간과 이징 곡선을 사용하는 좀 더 고급 예시입니다:

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

template

```
/*
  진입과 퇴장 애니메이션에
  서로 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;
}
```

CSS

Toggle Slide + Fade



▶ 플레이그라운드에서 실행해보기

## CSS 애니메이션

네이티브 CSS 애니메이션은 CSS 트랜지션과 동일한 방식으로 적용되지만, `*-enter-from` 이 요소가 삽입된 직후 바로 제거되는 것이 아니라 `animationend` 이벤트에서 제거된다는 차이점이 있습니다.

대부분의 CSS 애니메이션의 경우, `*-enter-active` 와 `*-leave-active` 클래스에 선언하면 됩니다. 예시는 다음과 같습니다:

```
<Transition name="bounce">
  <p v-if="show" style="text-align: center;">
    Hello here is some bouncy text!
  </p>
</Transition>
```

template

```
.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);
  }
}
```

CSS

Toggle

Hello here is some bouncy text!

▶ 플레이그라운드에서 실행해보기



다음과 같은 prop을 <Transition> 에 전달하여 커스텀 트랜지션 클래스를 지정할 수도 있습니다:

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

이들은 기존의 클래스 이름을 덮어씁니다. 이는 **Animate.css**와 같은 기존 CSS 애니메이션 라이브러리와 Vue의 트랜지션 시스템을 결합하고 싶을 때 특히 유용합니다:

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

template

▶ 플레이그라운드에서 실행해보기

## 트랜지션과 애니메이션을 함께 사용하기

Vue는 트랜지션이 끝났는지 알기 위해 이벤트 리스너를 부착해야 합니다. 적용된 CSS 규칙의 종류에 따라 `transitionend` 또는 `animationend` 가 될 수 있습니다. 둘 중 하나만 사용하는 경우, Vue가 자동으로 올바른 타입을 감지할 수 있습니다.

하지만, 같은 요소에 둘 다 사용하고 싶을 때가 있습니다. 예를 들어, Vue에 의해 트리거되는 CSS 애니메이션과, hover 시 CSS 트랜지션 효과를 함께 사용하고 싶을 때입니다. 이런 경우, Vue가 신경 써야 할 타입을 `type` prop을 통해 명시적으로 선언해야 하며, 값은 `animation` 또는 `transition` 중 하나입니다:

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

template

## 중첩 트랜지션과 명시적 트랜지션 지속 시간

트랜지션 클래스는 <Transition> 의 직접 자식 요소에만 적용되지만, 중첩된 CSS 선택자를 사용하여 중첩 요소에도 트랜지션을 적용할 수 있습니다:



```
<div v-if="show" class="outer">
  <div class="inner">
    Hello
  </div>
</div>
</Transition>
```

```
/* 중첩 요소를 타겟팅하는 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;
}

/* ... 필요한 다른 CSS는 생략 */
```

CSS

진입 시 중첩 요소에 트랜지션 지연을 추가하여, 계단식 진입 애니메이션 시퀀스를 만들 수도 있습니다:

```
/* 계단식 효과를 위해 중첩 요소의 진입을 지연 */
.nested-enter-active .inner {
  transition-delay: 0.25s;
}
```

CSS

하지만, 이로 인해 작은 문제가 발생합니다. 기본적으로 `<Transition>` 컴포넌트는 루트 트랜지션 요소에서 **첫 번째** `transitionend` 또는 `animationend` 이벤트를 감지하여 트랜지션이 끝났는지 자동으로 판단합니다. 중첩 트랜지션의 경우, 모든 내부 요소의 트랜지션이 끝날 때까지 기다리는 것이 바람직합니다.

이런 경우 `<Transition>` 컴포넌트의 `duration` prop을 사용하여 명시적으로 트랜지션 지속 시간(밀리초 단위)을 지정할 수 있습니다. 전체 지속 시간은 내부 요소의 지연 시간과 트랜지션 지속 시간을 합한 값이어야 합니다:

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

template

Toggle



Hello

### ▶ 플레이그라운드에서 실행해보기

필요하다면, 진입과 퇴장 지속 시간을 객체로 각각 지정할 수도 있습니다:

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

[template](#)

## 성능 고려사항

위에서 보여준 애니메이션들은 주로 `transform` 과 `opacity` 와 같은 속성을 사용합니다. 이 속성들은 애니메이션에 효율적인데, 그 이유는 다음과 같습니다:

1. 애니메이션 중 문서 레이아웃에 영향을 주지 않으므로, 매 프레임마다 비싼 CSS 레이아웃 계산이 발생하지 않습니다.
2. 대부분의 최신 브라우저는 `transform` 애니메이션 시 GPU 하드웨어 가속을 활용할 수 있습니다.

반면, `height` 나 `margin` 과 같은 속성은 CSS 레이아웃을 트리거하므로 애니메이션 비용이 훨씬 크며, 주의해서 사용해야 합니다.

---

## 자바스크립트 훅

`<Transition>` 컴포넌트에서 이벤트를 리스닝하여 트랜지션 과정에 자바스크립트로 개입할 수 있습니다:

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

[template](#)



```
<!-- ... -->
</Transition>
```

js

```
// 요소가 DOM에 삽입되기 전에 호출됩니다.
// 이곳에서 요소의 "enter-from" 상태를 설정할 수 있습니다.
function onBeforeEnter(el) {}

// 요소가 삽입된 한 프레임 후에 호출됩니다.
// 이곳에서 진입 애니메이션을 시작할 수 있습니다.
function onEnter(el, done) {
  // done 콜백을 호출하여 트랜지션 종료를 알립니다.
  // CSS와 함께 사용할 경우 선택 사항입니다.
  done()
}

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

// 진입 트랜지션이 완료되기 전에 취소되었을 때 호출됩니다.
function onEnterCancelled(el) {}

// 퇴장 후 전에 호출됩니다.
// 대부분의 경우 leave 후만 사용하면 됩니다.
function onBeforeLeave(el) {}

// 퇴장 트랜지션이 시작될 때 호출됩니다.
// 이곳에서 퇴장 애니메이션을 시작할 수 있습니다.
function onLeave(el, done) {
  // done 콜백을 호출하여 트랜지션 종료를 알립니다.
  // CSS와 함께 사용할 경우 선택 사항입니다.
  done()
}

// 퇴장 트랜지션이 끝나고
// 요소가 DOM에서 제거되었을 때 호출됩니다.
function onAfterLeave(el) {}

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

이 훅들은 CSS 트랜지션/애니메이션과 함께 또는 단독으로 사용할 수 있습니다.

자바스크립트 전용 트랜지션을 사용할 때는 `:css="false"` prop을 추가하는 것이 좋습니다. 이는 Vue에게 자동 CSS 트랜지션 감지를 건너뛰라고 명시적으로 알립니다. 약간 더 성능이 좋을 뿐만 아니라, CSS 규칙이 트랜지션에 실수로 간섭하는 것도 방지할 수 있습니다:

template

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



:css="false" 를 사용하면 트랜지션 종료 시점을 완전히 직접 제어해야 합니다. 이 경우, @enter 와 @leave 혹에서 done 콜백이 필수입니다. 그렇지 않으면 혹이 동기적으로 호출되어 트랜지션이 즉시 끝나게 됩니다.

아래는 **GSAP** 라이브러리를 사용하여 애니메이션을 수행하는 데모입니다. 물론 **Anime.js**나 **Motion One** 등 다른 애니메이션 라이브러리도 사용할 수 있습니다:

Toggle

▶ 플레이그라운드에서 실행해보기

## 재사용 가능한 트랜지션

트랜지션은 Vue의 컴포넌트 시스템을 통해 재사용할 수 있습니다. 재사용 가능한 트랜지션을 만들기 위해, <Transition> 컴포넌트를 감싸고 슬롯 콘텐츠를 전달하는 컴포넌트를 만들 수 있습니다:

MyTransition.vue

```
<script>
// 자바스크립트 혹 로직...
</script>

<template>
  <!-- 내장 Transition 컴포넌트를 감쌉니다 -->
  <Transition
    name="my-transition"
    @enter="onEnter"
    @leave="onLeave">
    <slot></slot> <!-- 슬롯 콘텐츠 전달 -->
  </Transition>
</template>

<style>
/*
  필요한 CSS...
  참고: 여기서 <style scoped>를 사용하지 마세요.
  슬롯 콘텐츠에는 적용되지 않습니다.
```



이제 `MyTransition` 을 내장 버전처럼 import하여 사용할 수 있습니다:

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

template

## 등장 시 트랜지션

노드의 초기 렌더링 시에도 트랜지션을 적용하고 싶다면, `appear` prop을 추가할 수 있습니다:

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

template

## 요소 간 트랜지션

`v-if` / `v-show` 로 요소를 토글하는 것 외에도, `v-if` / `v-else` / `v-else-if` 를 사용하여 두 요소 간에 트랜지션할 수 있습니다. 단, 한 번에 하나의 요소만 표시되도록 해야 합니다:

```
<Transition>
  <button v-if="docState === 'saved'">Edit</button>
  <button v-else-if="docState === 'edited'">Save</button>
  <button v-else-if="docState === 'editing'">Cancel</button>
</Transition>
```

template

Click to cycle through states:

Edit

 [플레이그라운드에서 실행해보기](#)

## 트랜지션 모드



할 때 레이아웃 문제를 피하기 위해 `position: absolute` 를 사용해야 했습니다.

하지만, 어떤 경우에는 이것이 불가능하거나 원하지 않는 동작일 수 있습니다. 퇴장 요소가 먼저 애니메이션되고, 진입 요소는 퇴장 애니메이션이 끝난 **후**에만 삽입되길 원할 수 있습니다. 이런 애니메이션을 수동으로 조율하는 것은 매우 복잡하지만, `<Transition>` 에 `mode prop`을 전달하여 이 동작을 쉽게 활성화할 수 있습니다:

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

template

아래는 `mode="out-in"` 을 적용한 이전 데모입니다:

Click to cycle through states:

Edit

`<Transition>` 은 `mode="in-out"` 도 지원하지만, 훨씬 덜 자주 사용됩니다.

---

## 컴포넌트 간 트랜지션

`<Transition>` 은 동적 컴포넌트에도 사용할 수 있습니다:

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

template

☒ A    ☐ B

Component A

 플레이그라운드에서 실행해보기

---

## 동적 트랜지션



라 서로 다른 트랜지션을 동적으로 적용할 수 있습니다:

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

template

Vue의 트랜지션 클래스 규칙을 사용해 CSS 트랜지션/애니메이션을 정의하고, 이를 전환하고 싶을 때 유용합니다.

또한, 컴포넌트의 현재 상태에 따라 자바스크립트 트랜지션 혹에서 서로 다른 동작을 적용할 수도 있습니다. 마지막으로, 재사용 가능한 트랜지션 컴포넌트를 통해 prop을 받아 트랜지션의 성격을 바꿀 수도 있습니다. 다소 진부하게 들릴 수 있지만, 정말로 상상력이 한계입니다.

---

## key 속성을 사용한 트랜지션

때로는 트랜지션이 발생하도록 DOM 요소의 리렌더를 강제로 해야 할 필요가 있습니다.

예를 들어, 다음 카운터 컴포넌트를 보세요:

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

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

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

vue

key 속성을 제외했다면, 텍스트 노드만 업데이트되어 트랜지션이 발생하지 않습니다. 하지만 key 속성이 있으면, count 가 변경될 때마다 Vue는 새로운 span 요소를 생성하므로 Transition 컴포넌트가 트랜지션할 두 개의 서로 다른 요소를 갖게 됩니다.

 [플레이그라운드에서 실행해보기](#)



 [GitHub에서 이 페이지 편집](#)

---

[< Previous](#)

플러그인

[Next >](#)

TransitionGroup