> ! You're browsing the documentation for v2.x and earlier. For v3.x, **click here [https://v3.vuejs.org/]** .

Enter/Leave & List Transitions

Overview [#Overview]

Vue provides a variety of ways to apply transition effects when items are inserted, updated, or removed from the DOM. This includes tools to:

- automatically apply classes for CSS transitions and animations
- integrate 3rd-party CSS animation libraries, such as Animate.css
- use JavaScript to directly manipulate the DOM during transition hooks
- integrate 3rd-party JavaScript animation libraries, such as Velocity.js

On this page, we'll only cover entering, leaving, and list transitions, but you can see the next section for **managing state transitions [transitioning-state.html]** .

Transitioning Single Elements/Components [#Transitioning-Single-Elements-Components]

Vue provides a `transition` wrapper component, allowing you to add entering/leaving transitions for any element or component in the following contexts:

- Conditional rendering (using `v-if`)
- Conditional display (using `v-show`)
- Dynamic components
- Component root nodes

This is what an example looks like in action:

HTML

```
<div id="demo">
  <button v-on:click="show = !show">
    Toggle
  </button>
  <transition name="fade">
    <p v-if="show">hello</p>
  </transition>
</div>
```

JS

```
new Vue({
  el: '#demo',
  data: {
    show: true
  }
})
```

CSS

```
.fade-enter-active, .fade-leave-active {
  transition: opacity .5s;
}
.fade-enter, .fade-leave-to /* .fade-leave-active below version 2.1.8 */
  opacity: 0;
}
```

Toggle
hello

When an element wrapped in a `transition` component is inserted or removed, this is what happens:

1. Vue will automatically sniff whether the target element has CSS transitions or animations applied. If it does, CSS transition classes will be added/removed at appropriate timings.

2. If the transition component provided **JavaScript hooks [#JavaScript-Hooks]** , these hooks will be called at appropriate timings.

3. If no CSS transitions/animations are detected and no JavaScript hooks are provided, the DOM operations for insertion and/or removal will be executed immediately on next frame (Note: this is a browser animation frame, different from Vue's concept of `nextTick`).

Transition Classes [#Transition-Classes]

There are six classes applied for enter/leave transitions.

1. `v-enter`: Starting state for enter. Added before element is inserted, removed one frame after element is inserted.

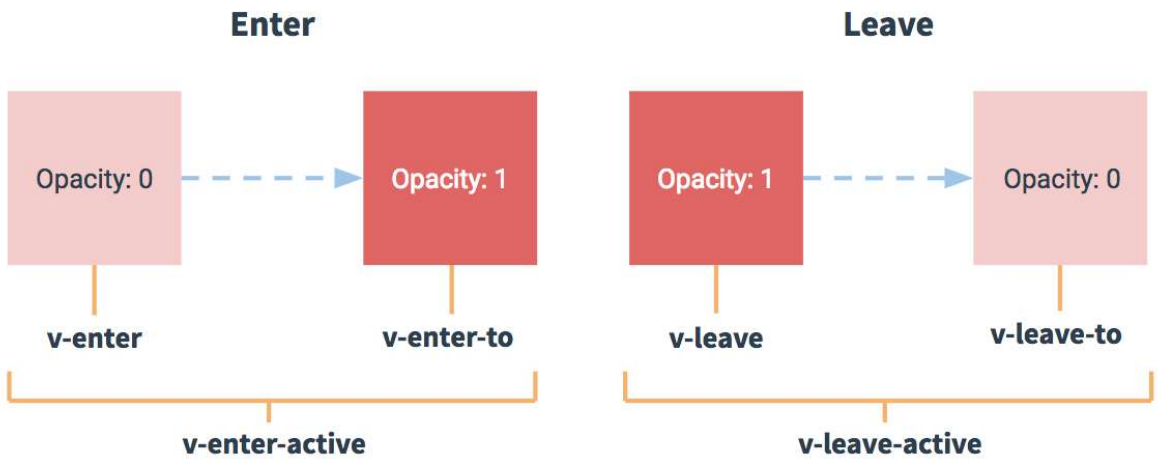
2. `v-enter-active`: Active state for enter. Applied during the entire entering phase. Added before element is inserted, removed when transition/animation finishes. This class can be used to define the duration, delay and easing curve for the entering transition.

3. `v-enter-to`: **Only available in versions 2.1.8+.** Ending state for enter. Added one frame after element is inserted (at the same time `v-enter` is removed), removed when transition/animation finishes.

4. `v-leave`: Starting state for leave. Added immediately when a leaving transition is triggered, removed after one frame.

5. `v-leave-active`: Active state for leave. Applied during the entire leaving phase. Added immediately when leave transition is triggered, removed when the transition/animation finishes. This class can be used to define the duration, delay and easing curve for the leaving transition.

6. `v-leave-to`: **Only available in versions 2.1.8+.** Ending state for leave. Added one frame after a leaving transition is triggered (at the same time `v-leave` is removed), removed when the transition/animation finishes.



Each of these classes will be prefixed with the name of the transition. Here the `v-` prefix is the default when you use a `<transition>` element with no name. If you use `<transition name="my-transition">` for example, then the `v-enter` class would instead be `my-transition-enter` .

`v-enter-active` and `v-leave-active` give you the ability to specify different easing curves for enter/leave transitions, which you'll see an example of in the following section.

CSS Transitions [#CSS-Transitions]

One of the most common transition types uses CSS transitions. Here's an example:

HTML

```
<div id="example-1">
  <button @click="show = !show">
    Toggle render
  </button>
  <transition name="slide-fade">
    <p v-if="show">hello</p>
  </transition>
</div>
```

JS

```
new Vue({
  el: '#example-1',
  data: {
```

```
    show: true
  }
})
```

CSS

```
/* Enter and leave animations can use different */
/* durations and timing functions.              */
.slide-fade-enter-active {
  transition: all .3s ease;
}
.slide-fade-leave-active {
  transition: all .8s cubic-bezier(1.0, 0.5, 0.8, 1.0);
}
.slide-fade-enter, .slide-fade-leave-to
/* .slide-fade-leave-active below version 2.1.8 */ {
  transform: translateX(10px);
  opacity: 0;
}
```

Toggle render

CSS Animations [#CSS-Animations]

CSS animations are applied in the same way as CSS transitions, the difference being that `v-enter` is not removed immediately after the element is inserted, but on an `animationend` event.

Here's an example, omitting prefixed CSS rules for the sake of brevity:

HTML

```
<div id="example-2">
  <button @click="show = !show">Toggle show</button>
  <transition name="bounce">
    <p v-if="show">Lorem ipsum dolor sit amet, consectetur adipiscing eli
  </transition>
</div>
```

JS

```
new Vue({
  el: '#example-2',
  data: {
    show: true
  }
})
```

CSS

```
.bounce-enter-active {
  animation: bounce-in .5s;
}
.bounce-leave-active {
  animation: bounce-in .5s reverse;
}
@keyframes bounce-in {
  0% {
    transform: scale(0);
  }
  50% {
    transform: scale(1.5);
  }
  100% {
    transform: scale(1);
  }
}
```

Toggle show

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris facilisis enim libero, at lacinia diam fermentum id. Pellentesque habitant morbi tristique senectus et netus.

Custom Transition Classes [#Custom-Transition-Classes]

You can also specify custom transition classes by providing the following attributes:

- `enter-class`
- `enter-active-class`

- `enter-to-class` (2.1.8+)
- `leave-class`
- `leave-active-class`
- `leave-to-class` (2.1.8+)

These will override the conventional class names. This is especially useful when you want to combine Vue's transition system with an existing CSS animation library, such as **Animate.css [https://daneden.github.io/animate.css/]** .

Here's an example:

HTML

```
<link href="https://cdn.jsdelivr.net/npm/animate.css@3.5.1" rel="styleshe

<div id="example-3">
  <button @click="show = !show">
    Toggle render
  </button>
  <transition
    name="custom-classes-transition"
    enter-active-class="animated tada"
    leave-active-class="animated bounceOutRight"
  >
    <p v-if="show">hello</p>
  </transition>
</div>
```

JS

```
new Vue({
  el: '#example-3',
  data: {
    show: true
  }
})
```

Toggle render

hello

Using Transitions and Animations Together [#Using-Transitions-and-Animations-Together]

Vue needs to attach event listeners in order to know when a transition has ended. It can either be `transitionend` or `animationend`, depending on the type of CSS rules applied. If you are only using one or the other, Vue can automatically detect the correct type.

However, in some cases you may want to have both on the same element, for example having a CSS animation triggered by Vue, along with a CSS transition effect on hover. In these cases, you will have to explicitly declare the type you want Vue to care about in a `type` attribute, with a value of either `animation` or `transition`.

Explicit Transition Durations [#Explicit-Transition-Durations]

> **New in 2.2.0+**

In most cases, Vue can automatically figure out when the transition has finished. By default, Vue waits for the first `transitionend` or `animationend` event on the root transition element. However, this may not always be desired - for example, we may have a choreographed transition sequence where some nested inner elements have a delayed transition or a longer transition duration than the root transition element.

In such cases you can specify an explicit transition duration (in milliseconds) using the `duration` prop on the `<transition>` component:

HTML

```
<transition :duration="1000">...</transition>
```

You can also specify separate values for enter and leave durations:

HTML

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

JavaScript Hooks [#JavaScript-Hooks]

You can also define JavaScript hooks in attributes:

HTML

```
<transition
  v-on:before-enter="beforeEnter"
  v-on:enter="enter"
  v-on:after-enter="afterEnter"
  v-on:enter-cancelled="enterCancelled"

  v-on:before-leave="beforeLeave"
  v-on:leave="leave"
  v-on:after-leave="afterLeave"
  v-on:leave-cancelled="leaveCancelled"
>
  <!-- ... -->
</transition>
```

JS

```
// ...
methods: {
  // --------
  // ENTERING
  // --------

  beforeEnter: function (el) {
    // ...
  },
  // the done callback is optional when
  // used in combination with CSS
  enter: function (el, done) {
    // ...
    done()
  },
  afterEnter: function (el) {
    // ...
  },
  enterCancelled: function (el) {
    // ...
  },

  // --------
```

```
    // LEAVING
    // --------

    beforeLeave: function (el) {
      // ...
    },
    // the done callback is optional when
    // used in combination with CSS
    leave: function (el, done) {
      // ...
      done()
    },
    afterLeave: function (el) {
      // ...
    },
    // leaveCancelled only available with v-show
    leaveCancelled: function (el) {
      // ...
    }
  }
```

These hooks can be used in combination with CSS transitions/animations or on their own.

> ! When using JavaScript-only transitions, **the `done` callbacks are required for the `enter` and `leave` hooks**. Otherwise, the hooks will be called synchronously and the transition will finish immediately.

> ! It's also a good idea to explicitly add `v-bind:css="false"` for JavaScript-only transitions so that Vue can skip the CSS detection. This also prevents CSS rules from accidentally interfering with the transition.

Now let's dive into an example. Here's a JavaScript transition using Velocity.js:

HTML

```
<!--
Velocity works very much like jQuery.animate and is
a great option for JavaScript animations
```

```
-->
<script src="https://cdnjs.cloudflare.com/ajax/libs/velocity/1.2.3/velocity
```

```
<div id="example-4">
  <button @click="show = !show">
    Toggle
  </button>
  <transition
    v-on:before-enter="beforeEnter"
    v-on:enter="enter"
    v-on:leave="leave"
    v-bind:css="false"
  >
    <p v-if="show">
      Demo
    </p>
  </transition>
</div>
```

JS

```
new Vue({
  el: '#example-4',
  data: {
    show: false
  },
  methods: {
    beforeEnter: function (el) {
      el.style.opacity = 0
      el.style.transformOrigin = 'left'
    },
    enter: function (el, done) {
      Velocity(el, { opacity: 1, fontSize: '1.4em' }, { duration: 300 })
      Velocity(el, { fontSize: '1em' }, { complete: done })
    },
    leave: function (el, done) {
      Velocity(el, { translateX: '15px', rotateZ: '50deg' }, { duration:
      Velocity(el, { rotateZ: '100deg' }, { loop: 2 })
      Velocity(el, {
```

```
          rotateZ: '45deg',
          translateY: '30px',
          translateX: '30px',
          opacity: 0
        }, { complete: done })
      }
    }
  })
```

Toggle

Transitions on Initial Render [#Transitions-on-Initial-Render]

If you also want to apply a transition on the initial render of a node, you can add the `appear` attribute:

HTML

```
<transition appear>
  <!-- ... -->
</transition>
```

By default, this will use the transitions specified for entering and leaving. If you'd like however, you can also specify custom CSS classes:

HTML

```
<transition
  appear
  appear-class="custom-appear-class"
  appear-to-class="custom-appear-to-class" (2.1.8+)
  appear-active-class="custom-appear-active-class"
>
  <!-- ... -->
</transition>
```

and custom JavaScript hooks:

HTML

```
<transition
  appear
  v-on:before-appear="customBeforeAppearHook"
  v-on:appear="customAppearHook"
  v-on:after-appear="customAfterAppearHook"
  v-on:appear-cancelled="customAppearCancelledHook"
>
  <!-- ... -->
</transition>
```

In the example above, either `appear` attribute or `v-on:appear` hook will cause an appear transition.

Transitioning Between Elements [#Transitioning-Between-Elements]

We discuss **transitioning between components [#Transitioning-Between-Components]** later, but you can also transition between raw elements using `v-if` / `v-else`. One of the most common two-element transitions is between a list container and a message describing an empty list:

HTML

```
<transition>
  <table v-if="items.length > 0">
    <!-- ... -->
  </table>
  <p v-else>Sorry, no items found.</p>
</transition>
```

This works well, but there's one caveat to be aware of:

> ! When toggling between elements that have **the same tag name**, you must tell Vue that they are distinct elements by giving them unique `key` attributes. Otherwise, Vue's compiler will only replace the content of the element for efficiency. Even when technically unnecessary though, **it's considered good practice to always key multiple items within a `<transition>` component.**

For example:

HTML

```
<transition>
  <button v-if="isEditing" key="save">
    Save
  </button>
  <button v-else key="edit">
    Edit
  </button>
</transition>
```

In these cases, you can also use the `key` attribute to transition between different states of the same element. Instead of using `v-if` and `v-else`, the above example could be rewritten as:

HTML

```
<transition>
  <button v-bind:key="isEditing">
    {{ isEditing ? 'Save' : 'Edit' }}
  </button>
</transition>
```

It's actually possible to transition between any number of elements, either by using multiple `v-if`s or binding a single element to a dynamic property. For example:

HTML

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

Which could also be written as:

HTML

```
<transition>
  <button v-bind:key="docState">
    {{ buttonMessage }}
  </button>
</transition>
```

JS

```
// ...
computed: {
  buttonMessage: function () {
    switch (this.docState) {
      case 'saved': return 'Edit'
      case 'edited': return 'Save'
      case 'editing': return 'Cancel'
    }
  }
}
```

Transition Modes [#Transition-Modes]

There's still one problem though. Try clicking the button below:

off

As it's transitioning between the "on" button and the "off" button, both buttons are rendered - one transitioning out while the other transitions in. This is the default behavior of `<transition>` - entering and leaving happens simultaneously.

Sometimes this works great, like when transitioning items are absolutely positioned on top of each other:

off

And then maybe also translated so that they look like slide transitions:

off

Simultaneous entering and leaving transitions aren't always desirable though, so Vue offers some alternative **transition modes**:

- `in-out` : New element transitions in first, then when complete, the current element transitions out.
- `out-in` : Current element transitions out first, then when complete, the new element transitions in.

Now let's update the transition for our on/off buttons with `out-in` :

HTML

```
<transition name="fade" mode="out-in">
  <!-- ... the buttons ... -->
</transition>
```

off

With one attribute addition, we've fixed that original transition without having to add any special styling.

The `in-out` mode isn't used as often, but can sometimes be useful for a slightly different transition effect. Let's try combining it with the slide-fade transition we worked on earlier:

off

Pretty cool, right?

Transitioning Between Components [#Transitioning-Between-Components]

Transitioning between components is even simpler - we don't even need the `key` attribute. Instead, we wrap a **dynamic component [components.html#Dynamic-Components]** :

HTML

```
<transition name="component-fade" mode="out-in">
  <component v-bind:is="view"></component>
</transition>
```

JS

```
new Vue({
  el: '#transition-components-demo',
  data: {
    view: 'v-a'
  },
  components: {
    'v-a': {
      template: '<div>Component A</div>'
    },
    'v-b': {
      template: '<div>Component B</div>'
    }
  }
})
```

CSS

```
.component-fade-enter-active, .component-fade-leave-active {
  transition: opacity .3s ease;
}
.component-fade-enter, .component-fade-leave-to
/* .component-fade-leave-active below version 2.1.8 */ {
  opacity: 0;
}
```

◉ A ○ B
Component A

List Transitions [#List-Transitions]

So far, we've managed transitions for:

- Individual nodes
- Multiple nodes where only 1 is rendered at a time

So what about for when we have a whole list of items we want to render simultaneously, for example with `v-for`? In this case, we'll use the `<transition-group>` component. Before we dive into an example though, there are a few things that are important to know about this component:

- Unlike `<transition>`, it renders an actual element: a `<span>` by default. You can change the element that's rendered with the `tag` attribute.
- **Transition modes [#Transition-Modes]** are not available, because we are no longer alternating between mutually exclusive elements.
- Elements inside are **always required** to have a unique `key` attribute.
- CSS transition classes will be applied to inner elements and not to the group/container itself.

List Entering/Leaving Transitions [#List-Entering-Leaving-Transitions]

Now let's dive into an example, transitioning entering and leaving using the same CSS classes we've used previously:

HTML

```
<div id="list-demo">
  <button v-on:click="add">Add</button>
  <button v-on:click="remove">Remove</button>
  <transition-group name="list" tag="p">
    <span v-for="item in items" v-bind:key="item" class="list-item">
      {{ item }}
    </span>
  </transition-group>
</div>
```

JS

```
new Vue({
  el: '#list-demo',
  data: {
    items: [1,2,3,4,5,6,7,8,9],
    nextNum: 10
  },
  methods: {
    randomIndex: function () {
      return Math.floor(Math.random() * this.items.length)
```

```
    },
    add: function () {
      this.items.splice(this.randomIndex(), 0, this.nextNum++)
    },
    remove: function () {
      this.items.splice(this.randomIndex(), 1)
    },
  }
})
```

```css
.list-item {
  display: inline-block;
  margin-right: 10px;
}
.list-enter-active, .list-leave-active {
  transition: all 1s;
}
.list-enter, .list-leave-to /* .list-leave-active below version 2.1.8 */ {
  opacity: 0;
  transform: translateY(30px);
}
```

Add Remove
1 2 3 4 5 6 7 8 9

There's one problem with this example. When you add or remove an item, the ones around it instantly snap into their new place instead of smoothly transitioning. We'll fix that later.

List Move Transitions [#List-Move-Transitions]

The `<transition-group>` component has another trick up its sleeve. It can not only animate entering and leaving, but also changes in position. The only new concept you need to know to use this feature is the addition of **the `v-move` class**, which is added when items are changing positions. Like the other classes, its prefix will match the value of a

provided `name` attribute and you can also manually specify a class with the `move-class` attribute.

This class is mostly useful for specifying the transition timing and easing curve, as you'll see below:

HTML

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.14.1/loda

<div id="flip-list-demo" class="demo">
  <button v-on:click="shuffle">Shuffle</button>
  <transition-group name="flip-list" tag="ul">
    <li v-for="item in items" v-bind:key="item">
      {{ item }}
    </li>
  </transition-group>
</div>
```

JS

```
new Vue({
  el: '#flip-list-demo',
  data: {
    items: [1,2,3,4,5,6,7,8,9]
  },
  methods: {
    shuffle: function () {
      this.items = _.shuffle(this.items)
    }
  }
})
```

CSS

```
.flip-list-move {
  transition: transform 1s;
}
```

Shuffle

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

This might seem like magic, but under the hood, Vue is using an animation technique called **FLIP [https://aerotwist.com/blog/flip-your-animations/]** to smoothly transition elements from their old position to their new position using transforms.

We can combine this technique with our previous implementation to animate every possible change to our list!

HTML

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.14.1/loda

<div id="list-complete-demo" class="demo">
  <button v-on:click="shuffle">Shuffle</button>
  <button v-on:click="add">Add</button>
  <button v-on:click="remove">Remove</button>
  <transition-group name="list-complete" tag="p">
    <span
      v-for="item in items"
      v-bind:key="item"
      class="list-complete-item"
    >
      {{ item }}
    </span>
  </transition-group>
</div>
```

JS

```
new Vue({
  el: '#list-complete-demo',
  data: {
    items: [1,2,3,4,5,6,7,8,9],
    nextNum: 10
  },
  methods: {
    randomIndex: function () {
      return Math.floor(Math.random() * this.items.length)
    },
    add: function () {
      this.items.splice(this.randomIndex(), 0, this.nextNum++)
    },
    remove: function () {
      this.items.splice(this.randomIndex(), 1)
    },
    shuffle: function () {
      this.items = _.shuffle(this.items)
    }
  }
})
```

CSS

```
.list-complete-item {
  transition: all 1s;
  display: inline-block;
  margin-right: 10px;
}
.list-complete-enter, .list-complete-leave-to
/* .list-complete-leave-active below version 2.1.8 */ {
  opacity: 0;
  transform: translateY(30px);
}
.list-complete-leave-active {
  position: absolute;
}
```

Shuffle Add Remove

1 2 3 4 5 6 7 8 9

! One important note is that these FLIP transitions do not work with elements set to `display: inline` . As an alternative, you can use `display: inline-block` or place elements in a flex context.

These FLIP animations are also not limited to a single axis. Items in a multidimensional grid can be **transitioned too [https://codesandbox.io/s/github/vuejs/vuejs.org/tree/master/src/v2/examples/vue-20-list-move-transitions]** :

Lazy Sudoku

Keep hitting the shuffle button until you win.

Shuffle

1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9

Staggering List Transitions [#Staggering-List-Transitions]

By communicating with JavaScript transitions through data attributes, it’s also possible to stagger transitions in a list:

HTML

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/velocity/1.2.3/veloci

<div id="staggered-list-demo">
  <input v-model="query">
  <transition-group
```

```
    name="staggered-fade"
    tag="ul"
    v-bind:css="false"
    v-on:before-enter="beforeEnter"
    v-on:enter="enter"
    v-on:leave="leave"
  >
    <li
      v-for="(item, index) in computedList"
      v-bind:key="item.msg"
      v-bind:data-index="index"
    >{{ item.msg }}</li>
  </transition-group>
</div>
```

JS

```
new Vue({
  el: '#staggered-list-demo',
  data: {
    query: '',
    list: [
      { msg: 'Bruce Lee' },
      { msg: 'Jackie Chan' },
      { msg: 'Chuck Norris' },
      { msg: 'Jet Li' },
      { msg: 'Kung Fury' }
    ]
  },
  computed: {
    computedList: function () {
      var vm = this
      return this.list.filter(function (item) {
        return item.msg.toLowerCase().indexOf(vm.query.toLowerCase()) !==
      })
    }
  },
  methods: {
    beforeEnter: function (el) {
```

```
      el.style.opacity = 0
      el.style.height = 0
    },
    enter: function (el, done) {
      var delay = el.dataset.index * 150
      setTimeout(function () {
        Velocity(
          el,
          { opacity: 1, height: '1.6em' },
          { complete: done }
        )
      }, delay)
    },
    leave: function (el, done) {
      var delay = el.dataset.index * 150
      setTimeout(function () {
        Velocity(
          el,
          { opacity: 0, height: 0 },
          { complete: done }
        )
      }, delay)
    }
  }
})
```

- Bruce Lee
- Jackie Chan
- Chuck Norris
- Jet Li
- Kung Fury

Reusable Transitions [#Reusable-Transitions]

Transitions can be reused through Vue's component system. To create a reusable transition, all you have to do is place a `<transition>` or `<transition-group>` component at the root, then pass any children into the transition component.

Here's an example using a template component:

JS

```
Vue.component('my-special-transition', {
  template: '\
    <transition\
      name="very-special-transition"\
      mode="out-in"\
      v-on:before-enter="beforeEnter"\
      v-on:after-enter="afterEnter"\
    >\
      <slot></slot>\
    </transition>\
  ',
  methods: {
    beforeEnter: function (el) {
      // ...
    },
    afterEnter: function (el) {
      // ...
    }
  }
})
```

And **functional components [render-function.html#Functional-Components]** are especially well-suited to this task:

JS

```
Vue.component('my-special-transition', {
  functional: true,
  render: function (createElement, context) {
    var data = {
      props: {
        name: 'very-special-transition',
        mode: 'out-in'
      },
      on: {
```

```
        beforeEnter: function (el) {
          // ...
        },
        afterEnter: function (el) {
          // ...
        }
      }
    }
    return createElement('transition', data, context.children)
  }
})
```

Dynamic Transitions [#Dynamic-Transitions]

Yes, even transitions in Vue are data-driven! The most basic example of a dynamic transition binds the `name` attribute to a dynamic property.

HTML

```
<transition v-bind:name="transitionName">
  <!-- ... -->
</transition>
```

This can be useful when you've defined CSS transitions/animations using Vue's transition class conventions and want to switch between them.

Really though, any transition attribute can be dynamically bound. And it's not only attributes. Since event hooks are methods, they have access to any data in the context. That means depending on the state of your component, your JavaScript transitions can behave differently.

HTML

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/velocity/1.2.3/velocit

<div id="dynamic-fade-demo" class="demo">
  Fade In: <input type="range" v-model="fadeInDuration" min="0" v-bind:ma
  Fade Out: <input type="range" v-model="fadeOutDuration" min="0" v-bind:m
  <transition
    v-bind:css="false"
    v-on:before-enter="beforeEnter"
```

```
    v-on:enter="enter"
    v-on:leave="leave"
  >
    <p v-if="show">hello</p>
  </transition>
  <button
    v-if="stop"
    v-on:click="stop = false; show = false"
  >Start animating</button>
  <button
    v-else
    v-on:click="stop = true"
  >Stop it!</button>
</div>
```

JS

```
new Vue({
  el: '#dynamic-fade-demo',
  data: {
    show: true,
    fadeInDuration: 1000,
    fadeOutDuration: 1000,
    maxFadeDuration: 1500,
    stop: true
  },
  mounted: function () {
    this.show = false
  },
  methods: {
    beforeEnter: function (el) {
      el.style.opacity = 0
    },
    enter: function (el, done) {
      var vm = this
      Velocity(el,
        { opacity: 1 },
        {
          duration: this.fadeInDuration,
```

```
            complete: function () {
              done()
              if (!vm.stop) vm.show = false
            }
          }
        )
      },
      leave: function (el, done) {
        var vm = this
        Velocity(el,
          { opacity: 0 },
          {
            duration: this.fadeOutDuration,
            complete: function () {
              done()
              vm.show = true
            }
          }
        )
      }
    }
  })
```

Fade In: Fade Out:
hello
Start animating

Finally, the ultimate way of creating dynamic transitions is through components that accept props to change the nature of the transition(s) to be used. It may sound cheesy, but the only limit really is your imagination.

← Handling Edge Cases [/v2/guide/components-edge-cases.html]
State Transitions [/v2/guide/transitioning-state.html] →