Переходы и анимация

ПЕРЕХОДЫ

Vue располагает компонентом-обёрткой <transition>, позволяющим анимировать появление и исчезновение элемента или компонента в следующих случаях:

- Условная отрисовка (используя v-if)
- Условное отображение (используя v-show)
- Динамические компоненты
- Корневые элементы компонентов.

ПЕРЕХОДЫ

Рассмотрим пример:

```
<style>
        .fade-enter-active, .fade-leave-active {
            transition: opacity .5s;
        }
        .fade-enter, .fade-leave-to {
            opacity: 0;
        }
</style>
<div id="app">
    <button v-on:click="show = !show"> Переключить </button>
    <transition name="fade">
        <p v-if="show">привет</p>
    </transition>
</div>
<script>
    const app = new Vue({
        el: '#app',
        data: { show: true }
    });
</script>
```

ПЕРЕХОДЫ

Когда элемент, завёрнутый в компонент transition, вставляется или удаляется, происходят следующие действия:

- Vue автоматически узнаёт, применены ли к целевому элементу CSS-переходы или анимации. Если да, соответствующие CSS-классы будут должным образом обновлены в нужные моменты времени.
- Если для компонента указаны хуки JavaScript, они будут вызваны в соответствующие моменты времени.
- Если не обнаружено ни CSS-переходов и анимаций, ни JavaScript-хуков, операции DOM для вставки или удаления элемента будут выполнены непосредственно в следующем анимационном фрейме (т.е. мгновенно).

КЛАССЫ ПЕРЕХОДОВ

Есть шесть классов, применяющихся в анимациях появления и исчезновения элементов:

- v-enter: Означает начало анимации появления элемента. Этот класс добавляется перед вставкой элемента, а удаляется в следующем фрейме после вставки.
- v-enter-active: Означает, что анимация появления элемента активна. Этот класс остаётся в течение всей анимации. Он добавляется перед вставкой элемента, а удаляется как только переход или анимация прекратились. Используйте его для установки длительности, задержки и плавности (easing curve) анимации появления.

КЛАССЫ ПЕРЕХОДОВ

Есть шесть классов, применяющихся в анимациях появления и исчезновения элементов:

- v-enter-to: Доступно только в версиях 2.1.8+. Означает, что анимация появления элемента завершается. Класс добавляется в следующем фрейме после вставки элемента (тогда же, когда будет удалён v-enter), удаляется после завершения перехода или анимации.
- v-leave: Означает начало анимации исчезновения элемента. Класс добавляется как только началась анимация исчезновения, а удаляется в следующем фрейме после этого.

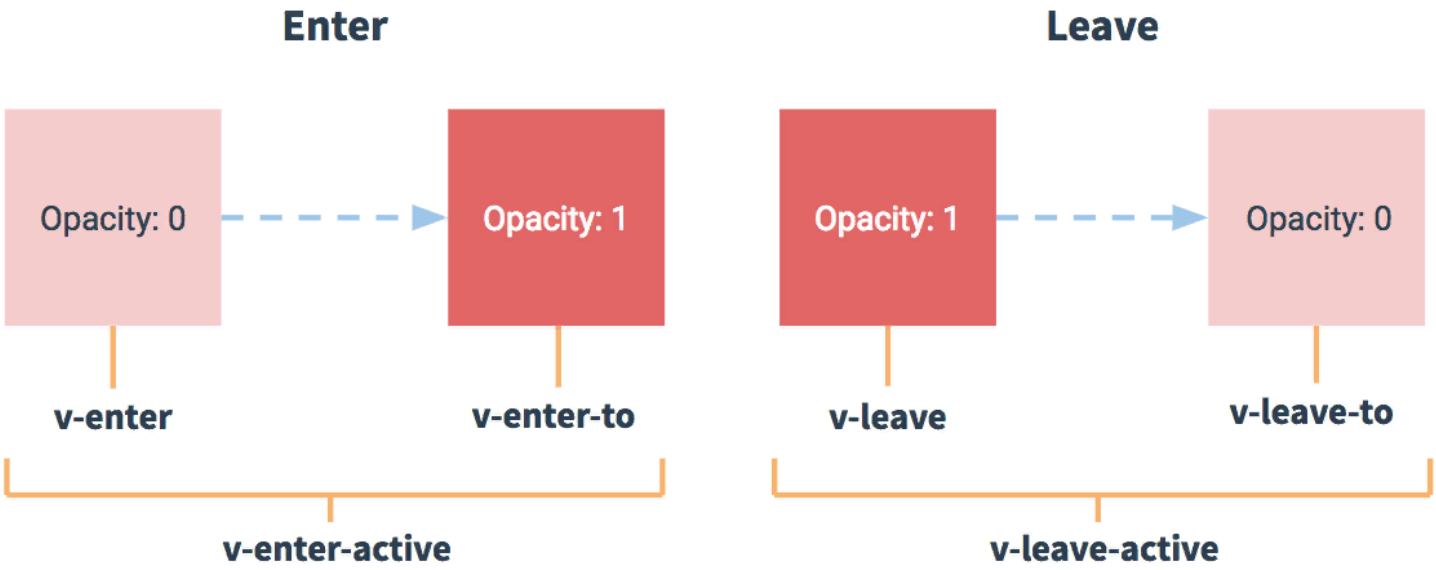
КЛАССЫ ПЕРЕХОДОВ

Есть шесть классов, применяющихся в анимациях появления и исчезновения элементов:

- v-leave-active: Означает, что анимация исчезновения элемента активна. Этот класс остаётся в течение всей анимации. Он добавляется как только начинается анимация исчезновения, а удаляется как только переход или анимация прекратились. Используйте его для установки длительности, задержки и плавности (easing curve) анимации исчезновения.

- v-leave-to: Доступно только в версиях 2.1.8+. Означает, что анимация исчезновения элемента завершается. Класс добавляется в следующем фрейме после начала анимации исчезновения (тогда же, когда будет удалён v-leave), а удаляется после завершения перехода или анимации.

КЛАССЫ ПЕРЕХОДОВ



Для каждого из этих классов указывается префикс, соответствующий имени перехода. Префикс v- применяется по умолчанию, если элемент <transition> используется без указания параметра name . Например, для <transition name="my-transition"> вместо класса v-enter будет применяться my-transition-enter.

v-enter-active и v-leave-active дают возможность указать различные анимационные эффекты для переходов появления и исчезновения элемента.

CSS-ПЕРЕХОДЫ

Одни из самых распространённых типов анимации — это CSS-переходы. Рассмотрим пример:

```
<style>
        .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 {
          transform: translateX(10px);
          opacity: 0;
        }
</style>
<div id="app">
    <button @click="show = !show"> Переключить </button>
    <transition name="slide-fade">
        <p v-if="show">привет</p>
    </transition>
</div>
```

CSS-АНИМАЦИИ

CSS-анимации применяются таким же образом, как и CSS-переходы, с одним отличием: v-enter удаляется не сразу же после вставки элемента, а только при наступлении события animationend. Рассмотрим пример:

```
<style>
      .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); }
       }
</style>
<div id="app">
   <button @click="show = !show"> Переключить </button>
   <transition name="bounce">
      <p v-if="show">Много текста …</p>
   </transition>
</div>
```

JAVASCRIPT-ХУКИ

Для более сложных переходов и анимации предусмотрены специальные хуки, которые можно объединять с кодом на JavaScript для манипуляции
и корректирования CSS-стилей. Рассмотрим пример:

```
<style>
      @keyframes bounce-in {
          0% { transform: scale(0); }
          50% { transform: scale(1.5); }
          100% { transform: scale(1); }
       }
</style>
<div id="app">
   <button @click="show = !show"> Переключить </button>
   <transition @before-enter="beforeEnter" @enter="enter" @after-enter="afterEnter"
      @enter-cancelled="enterCancelled" @before-leave="beforeLeave" @leave="leave"
      @after-leave="afterLeave" @leave-cancelled="leaveCancelled" :css="false">
      <p v-if="show">Много текста …</p>
   </transition>
</div>
<script>
   function addEventListener(el, done) {
      el.addEventListener("animationend", function() {
         el.style="";
         done();
      });
   }
```

JAVASCRIPT-ХУКИ

Пример (продолжение):

```
    const app = new Vue({
        el: '#app',
        data: { show: true },
        methods: {
            enter: function(el, done) {
                console.log("enter");
                addEventListener(el, done);
                el.style.animation = "bounce-in 3s";
            },
            leave: function(el, done) {
                console.log("leave");
                addEventListener(el, done);
                el.style.animation = "bounce-in 3s reverse";
            },
            beforeEnter: function(el) { console.log("before enter"); },
            afterEnter: function(el) { console.log("after enter"); },
            enterCancelled: function(el) { console.log("enter cancelled"); },
            beforeLeave: function(el) { console.log("before leave"); },
            afterLeave: function(el) { console.log("after leave"); },
            // leaveCancelled доступна только для v-show
            leaveCancelled: function(el) { console.log("leave cancelled"); }
        }
    });
</script>
```

ПЕРЕХОДЫ МЕЖДУ ЭЛЕМЕНТАМИ

Рассмотрим пример:

```
<style>
        .fade-enter-active, .fade-leave-active {
            transition: opacity .5s
        }
        .fade-enter, .fade-leave-active {
            opacity: 0
        }
</style>
<div id="app">
 <button @click="isEditing = !isEditing"> Переключить </button>
    <transition name="fade">
        <button v-if="isEditing">Сохранить</button>
        <button v-else>Редактировать</button>
    </transition>
</div>
<script>
    const app = new Vue({
        el: '#app',
        data: { isEditing: true }
    });
</script>
```

В данном случае компилятор Vue из соображений эффективности только поменяет содержимое элемента при переключения и трансформация не сработает.

ПЕРЕХОДЫ МЕЖДУ ЭЛЕМЕНТАМИ

При переключении между элементами, использующими одноимённые теги, нужно указать Vue, что это различные элементы, установив уникальные значения атрибута key. Модифицируем пример:

```
<style>
        .fade-enter-active, .fade-leave-active {
            transition: opacity .5s
        }
        .fade-enter, .fade-leave-active {
            opacity: 0
        }
</style>
<div id="app">
 <button @click="isEditing = !isEditing"> Переключить </button>
    <transition name="fade">
        <button v-if="isEditing" key="save">Сохранить</button>
        <button v-else key="edit">Редактировать</button>
    </transition>
</div>
<script>
    const app = new Vue({
        el: '#app',
        data: { isEditing: true }
    });
</script>
```

РЕЖИМЫ ПЕРЕХОДОВ

Модифицируем предыдущий пример:

```
<style>
        .absolute-wrapper {
          position: relative;
          height: 18px;
        }
        .absolute-wrapper button {
          position: absolute;
        }
        .fade-enter-active, .fade-leave-active {
            transition: opacity .5s
        }
        .fade-enter, .fade-leave-active {
            opacity: 0
        }
</style>
<div id="app">
 <button @click="isEditing = !isEditing"> Переключить </button>
 <div class="absolute-wrapper">
    <transition name="fade">
        <button v-if="isEditing" key="save">Сохранить</button>
        <button v-else key="edit">Редактировать</button>
    </transition>
  </div>
</div>
```

В предыдущем примере во время перехода от кнопки «Сохранить» к кнопке «Редактировать» одновременно отображаются обе кнопки: одна — исчезая, другая — появляясь. Так <transition> ведёт себя по умолчанию. Иногда это поведение подходит, например если оба элемента абсолютно спозиционированы в одном и том же месте.

РЕЖИМЫ ПЕРЕХОДОВ

Таким образом можно также сымитировать анимацию слайдера:

```
<style>
        .translate-wrapper { position: relative; height: 18px; }
        .translate-wrapper button { position: absolute; }
        .translate-fade-enter-active, .translate-fade-leave-active {
          transition: all 1s;
        }
        .translate-fade-enter, .translate-fade-leave-active { opacity: 0; }
        .translate-fade-enter {
          transform: translateX(31px);
        }
        .translate-fade-leave-active {
          transform: translateX(-31px);
        }
</style>
<div id="app">
 <button @click="isEditing = !isEditing"> Переключить </button>
 <div class="translate-wrapper">
    <transition name="translate-fade">
        <button v-if="isEditing" key="save">Сохранить</button>
        <button v-else key="edit">Редактировать</button>
    </transition>
  </div>
</div>
```

РЕЖИМЫ ПЕРЕХОДОВ

Тем не менее, одновременное сокрытие и появление элементов — это не всегда то, чего хочется. Поэтому Vue предоставляет альтернативные режимы перехода:

- in-out: Сначала появляется новый элемент, и только после этого исчезает старый.
- out-in: Сначала исчезает старый элемент, и только после этого появляется новый.

Давайте изменим переход для наших кнопок, используя out-in:

```
<div id="app">
 <button @click="isEditing = !isEditing"> Переключить </button>
 <div class="translate-wrapper">
    <transition name="translate-fade" mode="out-in" >
        <button v-if="isEditing" key="save">Сохранить</button>
        <button v-else key="edit">Редактировать</button>
    </transition>
  </div>
</div>
```

И используя in-out:

```
<div id="app">
 <button @click="isEditing = !isEditing"> Переключить </button>
 <div class="translate-wrapper">
    <transition name="translate-fade" mode="in-out" >
        <button v-if="isEditing" key="save">Сохранить</button>
        <button v-else key="edit">Редактировать</button>
    </transition>
  </div>
</div>
```

ПЕРЕХОДЫ ПРИ ПЕРВИЧНОЙ ОТРИСОВКЕ

Если вы хотите, чтобы пользователь увидел анимацию перехода и при изначальной отрисовке, добавьте атрибут appear. По умолчанию будут задействованы переходы, указанные для появления и исчезновения.

```
<div id="app">
 <button @click="isEditing = !isEditing"> Переключить </button>
 <div class="translate-wrapper">
   <transition name="translate-fade" mode="in-out" appear>
      <button v-if="isEditing" key="save">Сохранить</button>
      <button v-else key="edit">Редактировать</button>
   </transition>
 </div>
</div>
```

Можно, указать и отдельные классы первичной отрисовки:

```
<transition
  appear
  appear-class="custom-appear-class"
  appear-to-class="custom-appear-to-class"
  appear-active-class="custom-appear-active-class">
  <!-- ... -->
</transition>
```

ПЕРЕХОДЫ МЕЖДУ КОМПОНЕНТАМИ

Переходы между компонентами не требуют атрибут key.

```
 <style>
        .component-fade-enter-active, .component-fade-leave-active {
          transition: opacity .3s ease;
        }
        .component-fade-enter, .component-fade-leave-to { opacity: 0; }
</style>
<div id="app">
    <input v-model="view" type="radio" value="v-a" id="a" name="view"><label for="a">А</label>
    <input v-model="view" type="radio" value="v-b" id="b" name="view"><label for="b">Б</label>
    <transition name="component-fade" mode="out-in">
        <component v-bind:is="view"></component>
    </transition>
</div>
<script>
    const app = new Vue({
        el: '#app',
        data: { view: 'v-a' },
        components: {
            'v-a': { template: '<div>Компонент А</div>' },
            'v-b': { template: '<div>Компонент Б</div>' }
        }
    });
</script>
```

ПЕРЕХОД В СПИСКАХ

В случае, когда у нас есть целый список элементов, который мы бы хотели отображать одновременно, необходимо использовать компонент <transition-group>.
Особенности компонента <transition-group>:

- В отличие от <transition>, он создаёт реальный элемент. По умолчанию это , но можно изменить на любой другой, указав атрибут tag.
- Режимы переходов недоступны, так как мы больше не переключаемся туда-сюда между взаимоисключающими элементами.
- У каждого элемента внутри <transition-group> всегда обязательно должно быть уникальное значение атрибута key.
- CSS-классы переходов будут применяться к внутренним элементам, а не к самой группе/контейнеру.

ПЕРЕХОД В СПИСКАХ

Для анимации перемещения необходимо добавить класс v-move, который указывается при изменении позиции элементов в компоненте <transition-group>. Рассмотрим пример:

```
<style>
        .flip-list-move { transition: transform 1s; }
</style>
<div id="app">
    <button v-on:click="shuffle">Перемешать</button>
    <transition-group name="flip-list" tag="ul">
        <li v-for="item in items" :key="item">{{ item }}</li>
    </transition-group>
</div>
<script>
    new Vue({
        el: '#app',
        data: { items: [1,2,3,4,5,6,7,8,9] },
        methods: {
            shuffle: function () {
                this.items.sort(()=>{ return Math.random()*2 - 1; })
            }
        }
    })
</script>
```

!!!FLIP-анимации не работают с элементами, для которых установлен режим позиционирования display: inline. Вместо него используйте режим display: inline-block или flex-контейнер.

ПЕРЕХОД В СПИСКАХ

Рассмотрим пример анимации вставки и удаления элементов списка:

```
<style>
        .list-item {
            transition: all 1s;
            display: inline-block;
            margin-right: 10px;
        }
        .list-enter, .list-leave-to {
            opacity: 0;
            transform: translateY(30px);
        }
        .list-leave-active {
            position: absolute;
        }
</style>
<div id="app">
    <button v-on:click="add">Добавить</button>
    <button v-on:click="remove">Удалить</button>
    <transition-group name="list" tag="p">
        <span v-for="item in items" :key="item" class="list-item">{{ item }}</span>
    </transition-group>
</div>
```

ПЕРЕХОД В СПИСКАХ

Пример (продолжение):

```
<script>
    new Vue({
        el: '#app',
        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)
            },
        }
    })
</script>
```

ДОПОЛНИТЕЛЬНЫЕ ССЫЛКИ

Официальное руководство по Vue.js:
https://ru.vuejs.org/v2/guide/