Web Client - **#4**

[Vue.js - tools]

Cyril Rouyer

https://vuejs.org

Sommaire

- Mixins
- Directives
- Plugins
- Render functions
- Transitions & animations

Mixins

Mixins 1

Définition

- Permettent de partager du code à travers plusieurs composants

- Sous forme d’un objet

- Chaque propriété de l’objet correspond à une option potentielle d’un composant classique (methods, created, computed, …)

Mixins 2

Exemple

```
// définit un objet de mixin
var myMixin = {
  created: function () {
    this.hello()
  },
  methods: {
    hello: function () {
      console.log('hello from mixin!')
    }
  }
}

// définition d'un composant utilisant cette mixin
export default {
  ...
  mixins: [myMixin]
  ...
}
```

Mixins 3

Merge strategy

- Il se peut que les propriétés apportées par la mixin existent déjà dans le composant qui l'utilise

- Dans ce cas une stratégie simple s'applique au niveau de Vue :

- Les fonctions de hook (ex : created) seront stockées dans un tableau, celles des mixins en tête

- Les propriétés qui attendent un objet (ex : methods, computed) vont réaliser une fusion. Priorité donnée au composant pour les propriétés identiques

Mixins 4

Merge custom

- Bon à savoir :
  - Il est possible de surcharger la stratégie par défaut de Vue
  - Il faut modifier la config de Vue à travers **app.config.optionMergeStrategies**
  - Chaque propriété potentiellement présente dans vos mixins suit une stratégie par défaut définie dans cet objet lorsqu'un merge est nécessaire

Mixins 5

Merge custom

```
const app = createApp({
 msg: 'Vue',
 mixins: [
    {
      msg: 'Hello '
    }
  ],
  mounted() {
    console.log(this.$options.msg)
  }
})

app.config.optionMergeStrategies.msg = (parent,
child) => {
  return (parent || '') + (child || '')
}

app.mount(‘#app')
```

Mixins 6

Mixin globale

- On peut déclarer une mixin de façon globale, à l'échelle de Vue :

const app = createApp(App)

app.mixin({ … })

- Ce qui permet de diffuser en une fois le code réutilisable sur toutes les instances de Vue ou de librairies tierces

- Mais c'est rarement utilisé, parce que l'impact est trop large et induit des comportements qu'on ne maîtrise pas systématiquement

Mixins 7

Mixins locales

- À l'inverse on peut déclarer localement les mixins qui nous intéressent au sein d'un composant avec la propriété **mixins**

```
…,
mixins: [authMixin, userMixin],
…
```

Directives

Directives 1

Définition

- Vue.js apporte plusieurs directives : v-model, v-show, …
- Une directive s’applique sur un élément du DOM
- On peut créer nos propres directives
- Deux façons d’enregistrer une directive :
  - Globale (disponible dans tous les templates)
  - Locale (déclarée dans un et un seul composant)

Directives 2

Inscription globale

const app = createApp(App)
app.directive(‘name-of-the-directive’, { … })

```
// Register a global custom directive called `v-focus`
app.directive(‘focus', {
  // When the bound element is inserted into the DOM...
  mounted: function (el) {
    // Focus the element
    el.focus()
  }
})
```

Directives 3

Inscription locale

Propriété “directives” dans la déclaration du composant

```
...,
directives: {
  focus: {
    // directive definition
    mounted: function (el) {
      el.focus()
    }
  }
},
...
```

Directives 4

Utilisation

<input v-focus>

- Le nom de la directive est préfixé de ‘v-‘
- Le tout en kebab-case

Directives 5

Hooks

- **created** : appelé avant que les événements ou autres attributs de l'élément ne soient interprétés
- **beforeMount** : juste avant que l'élément ne soit inséré dans le DOM
- **mounted** : appelé quand le composant père et tous ses fils sont insérés
- **beforeUpdate** : avant que le composant parent ne soit mis à jour
- **updated** : après que le composant parent et ses fils n'aient été mis à jour
- **beforeUnmount** : avant que le composant parent ne soit démonté
- **unmounted** : quand le composant parent est démonté

Directives 6

Argument des Hooks

- Chaque hook est une fonction qui prend potentiellement 4 arguments :
  - el : l'élément du DOM sur lequel est attachée la directive
  - binding : un objet qui apporte des informations sur le contexte du binding entre l'élément et la directive
  - vnode : le noeud virtuel produit par le compilateur
  - prevVnode : le noeud virtuel précédent (avant changement). Cet argument ne sera disponible que dans le cas des hooks **beforeUpdate** et **updated**

Directives 7

Hooks - binding object

- L’argument binding contient toutes les informations liées au contexte d’attachement de la directive à l’élément

- Il s’agit d’un objet qui contient les propriétés suivantes :
  - **value** : la valeur de la directive (interprétée : 1 + 1 -> 2)
  - **oldValue** : la valeur avant update
  - **arg** : l’argument touché (v-mydir:href=“…” —> href)
  - **modifiers** : les modificateurs associés (v-mydir.foo.bar —> {foo: true, bar: true}
  - **dir :** l’objet représentant la définition de la directive

Directives 8

Hooks - version abrégée

- La plupart du temps on n’utilise que les hooks **mounted** et **updated** et le code est souvent le même

- Vue.js prévoit une version abrégée de la déclaration de la directive, en passant directement une fonction, qui servira aussi bien au hook **mounted** qu’au hook **updated**

```
app.directive(‘color-swatch', function (el, binding) {
  el.style.backgroundColor = binding.value
})
```

Directives 9

Objets littéraux

- Si la directive a besoin de plusieurs valeurs, on peut passer par un objet littéral

```
<div v-demo="{ color: 'white', text: 'bonjour !' }"></div>
```

```
app.directive(‘demo', function (el, binding) {
  console.log(binding.value.color) // => "white"
  console.log(binding.value.text)  // => "bonjour !"
})
```

Plugins

Plugins 1

Définition

- Un plugin est un set de surcharges ou de fonctionnalités réutilisables et surtout distribuables (ex : librairies externes)

- s’ajoute au niveau des fonctionnalités globales de votre instance de Vue

- Permet d’apporter des directives, des mixins, …, des extensions à l’instance de Vue.

Plugins 2

Déclaration

- Le plugin est un simple objet qui expose une méthode **install**

- Install prend deux arguments :

  - **app :** représente l'instance de votre Vue

  - **options** : facultatif, permet de fournir des informations au plugin, comme de la configuration

Plugins 3

Exemple

```
MyPlugin.install = function (app, options) {
  // 1. ajouter une ressource globale
  app.directive('my-directive', {
    mounted (el, binding, vnode, oldVnode) { … }
    ...
  })

  // 2. injecter des options de composant
  app.mixin({
    created: function () { … }
    ...
  })

  // 3. ajouter une méthode d’instance ou une propriété
  app.config.globalProperties.$myMethod = function (methodOptions) { … }
}
```

Plugins 4

Utilisation

Sans option

```
app.use(MyPlugin)
```

Avec options

```
app.use(MyPlugin, {
  color: ‘red’,
  size: ’30’
})
```

Render functions

Render functions 1

Définition

- Dans certains cas précis, les templates peuvent s'avérer compliqués,
- En particulier si l'élément HTML dépend d'un modèle
- Exemple : h1, h2, h… comment faire si on veut rendre un heading dont la taille dépend d'un modèle ?

Render functions 2

Sans fonction de rendu

```
<script type="text/x-template" id="anchored-heading-template">
  <h1 v-if="level === 1">
    <slot></slot>
  </h1>
  <h2 v-else-if="level === 2">
    <slot></slot>
  </h2>
  <h3 v-else-if="level === 3">
    <slot></slot>
  </h3>
  <h4 v-else-if="level === 4">
    <slot></slot>
  </h4>
  <h5 v-else-if="level === 5">
    <slot></slot>
  </h5>
  <h6 v-else-if="level === 6">
    <slot></slot>
  </h6>
</script>
```

```
<anchored-heading :level="1">Hello world!
</anchored-heading>
```

```
app.component('anchored-heading', {
  template: '#anchored-heading-template',
  props: {
    level: {
      type: Number,
      required: true
    }
  }
})
```

BEURK !

Render functions 3

Définition 2

- Le but des fonctions de rendu est d'écrire le template directement en Javascript
- En manipulant les VNodes
- Avec la fonction **h**(elemName, attrs, innerHTML)

Render functions 4

Avec une fonction de rendu

```
Import { h } from ‘vue’
app.component(‘anchored-heading', {
  render: function () {
    return h(
      'h' + this.level,   // nom de balise
      {id: “monSuperID”},
      this.$slots // le contenu
    )
  },
  props: {
    level: {
      type: Number,
      required: true
    }
  }
})
```

```
<anchored-heading :level="1">Hello world!
</anchored-heading>
```

Render functions 5

Zoom sur h

- h permet de créer des VNodes (élément du DOM Virtuel)
- Trois paramètres :
  - Element : string, objet ou fonction retournant l'un des deux précédents (ex : 'div')
  - Config : objet permettant de détailler le VNode (style, classes, id, directives, props, …)
  - Enfants : le contenu de l'élément courant, sous forme de VNode obtenu avec h, ou du texte pour de l'inner-text.
- Tout le détail ici : https://vuejs.org/api/render-function.html#render-function-apis

Transitions & Animations

Transitions & Animations 1

Transition d’entrée et de sortie

- Il est parfois nécessaire d’animer nos vues pour apporter un meilleur feedback à l’utilisateur

- Plusieurs effets peuvent ainsi être utilisés, soit en CSS directement, soit avec du JS

- Vue.js intègre des outils permettant de faciliter l’apparition ou la suppression d’éléments dans une liste, ou dans l’affichage conditionnel d’un élément

Transitions & Animations 2

<transition>

- Vue fournit un composant nommé “transition” qui permet d’encapsuler un élément qui doit être animé

- Le composant <transition> prend une propriété name qui définit la classe d’animation

- Dès que le contenu de template est affiché ou masqué (création du node, v-if, v-show), la classe correspondante est associée à l’élément

Transitions & Animations 3

<transition>

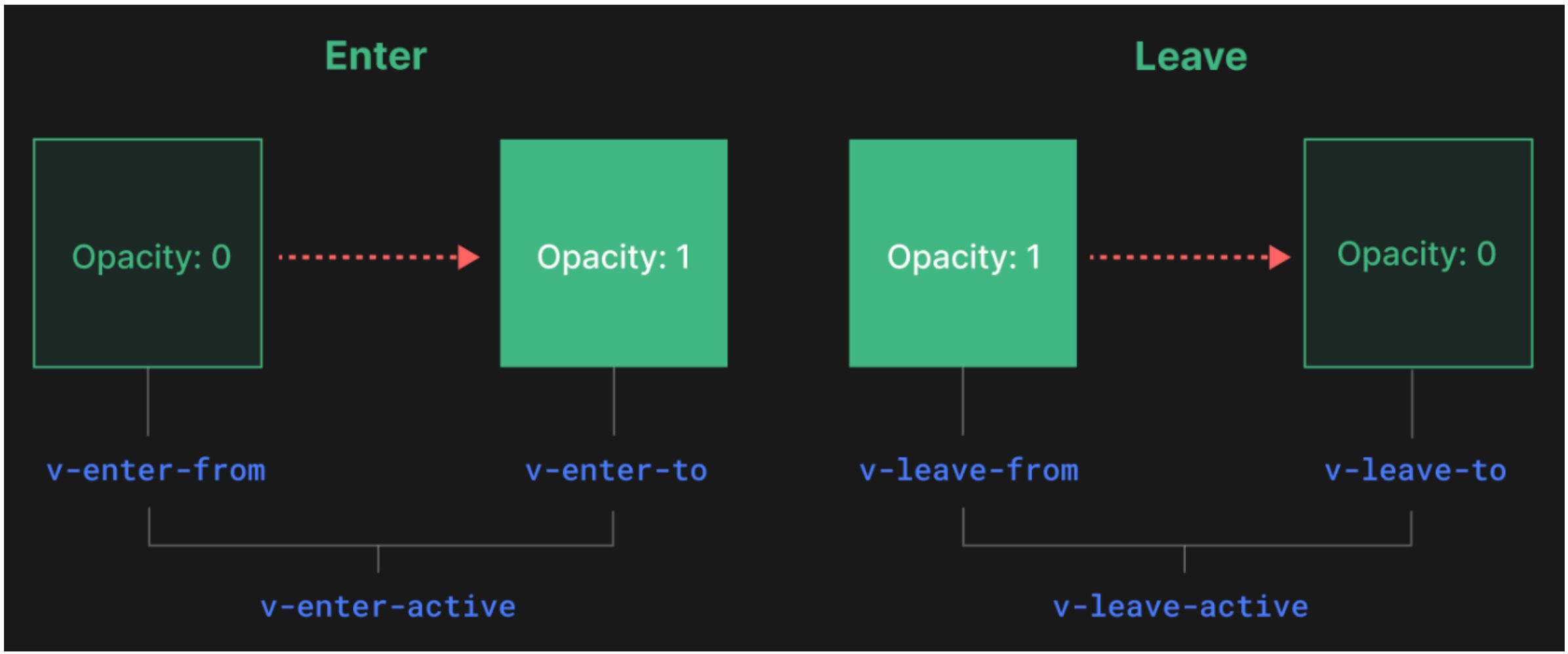
```
<div id="demo">
  <button v-on:click="show = !show">
    Permuter
  </button>
  <transition name="fade">
    <p v-if="show">bonjour</p>
  </transition>
</div>
```

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

```
.fade-enter-active, .fade-leave-active {
  transition: opacity .5s ease
}
.fade-enter-from, .fade-leave-to {
  opacity: 0
}
```

Transitions & Animations 4

Classes de transition



https://vuejs.org/guide/built-ins/transition.html#transition

Transitions & Animations 5

Classes de transition

- Si le composant <transition> n'est pas nommé, les classes par défaut commencent par v- (ex : v-enter, v-leave)

- Sinon, par le nom de la transition : <transition name="fade"> —> fade-enter-from, fade-leave-to

Transitions & Animations 6

Surcharge des classes de transition

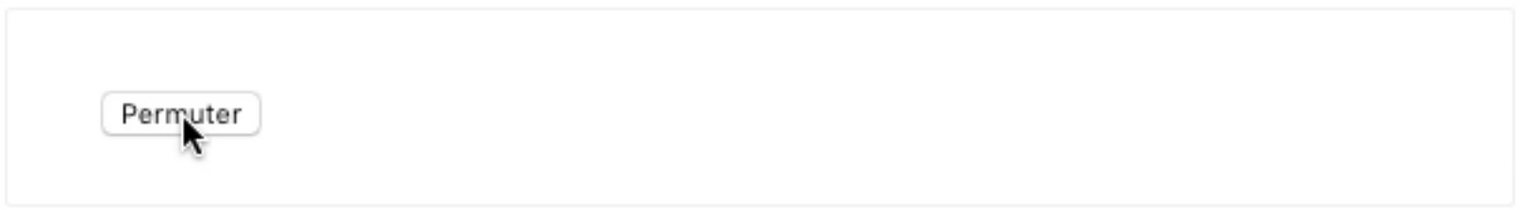
- On peut surcharger les classes de transition
- Pratique pour pouvoir utiliser les noms de classes d’une librairie externe (ex : Animate.css)
- Chaque classe peut être surchargée via une prop (enter-from-class, leave-from-class, enter-to-class, leave-to-class, enter-active-class, leave-active-class)

Transitions & Animations 7

Surcharge des classes de transition

```
<link href="https://cdn.jsdelivr.net/npm/animate.css@3.5.1" rel="stylesheet" type="text/css">

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



Transitions & Animations 8

Durée explicite

- On peut passer une propriété duration au composant pour forcer la durée de la transition/animation (en millisecondes)
- <transition :duration="1000">…</transition>
- <transition :duration="{enter: 500, leave: 800}">…</transition>

Transitions & Animations 9

Hooks Javascript

- Si on utilise une lib JS pour réaliser des animations (ex : Velocity.js), Vue.js permet de spécifier quelle méthode appeler dans chaque situation :

```
<transition
  @before-enter=“beforeEnter"
  @enter=“enter"
  @after-enter=“afterEnter"
  @enter-cancelled=“enterCancelled"

  @before-leave=“beforeLeave"
  @leave=“leave"
  @after-leave=“afterLeave"
  @leave-cancelled=“leaveCancelled"
>
  <!-- ... -->
</transition>
```

Transitions & Animations 10

Appear

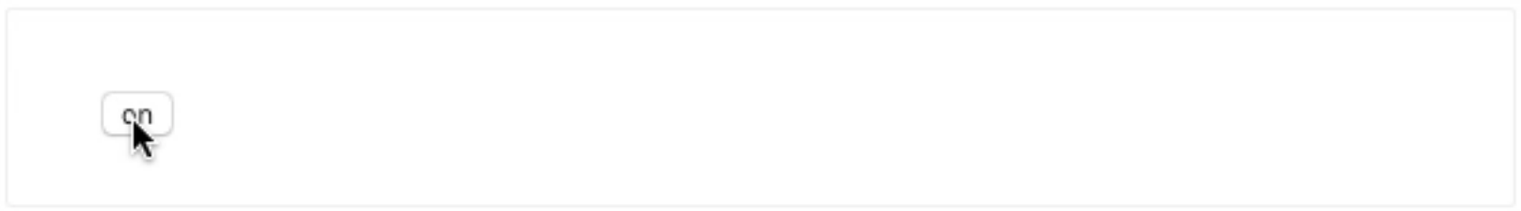
- On peut déclencher une animation lors du rendu initial d’un noeud

- Pour cela le composant transition prévoit une propriété **appear** (<transition appear>…</transition>

Transitions & Animations 11

Affichage conditionnel & mode

- Le composant <transition> tient compte de l'affichage conditionnel de son contenu (v-if, v-else, key)
- Mais il gère par défaut les transitions enter et leave en même temps ce qui peut avoir des effets indésirables



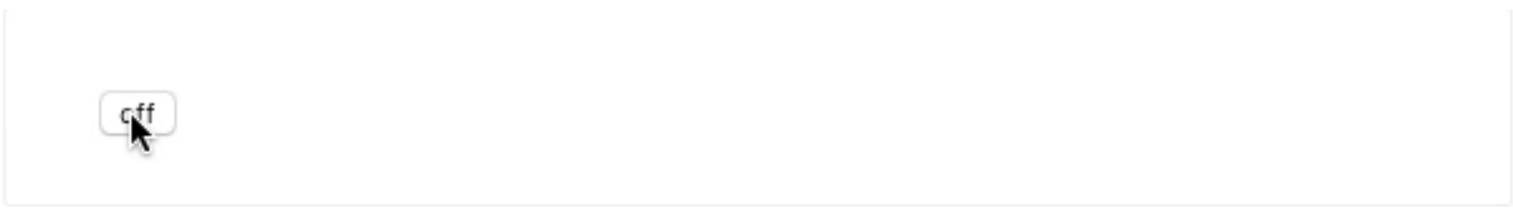
Transitions & Animations 12

Affichage conditionnel & mode

- Pour contourner ce souci, le composant <transition> peut prendre un mode qui permet d'orchestrer l'effet.

- Le mode peut être soit **in-out**, soit **out-in**

- <transition mode="in-out">…</transition>



Transitions & Animations 13

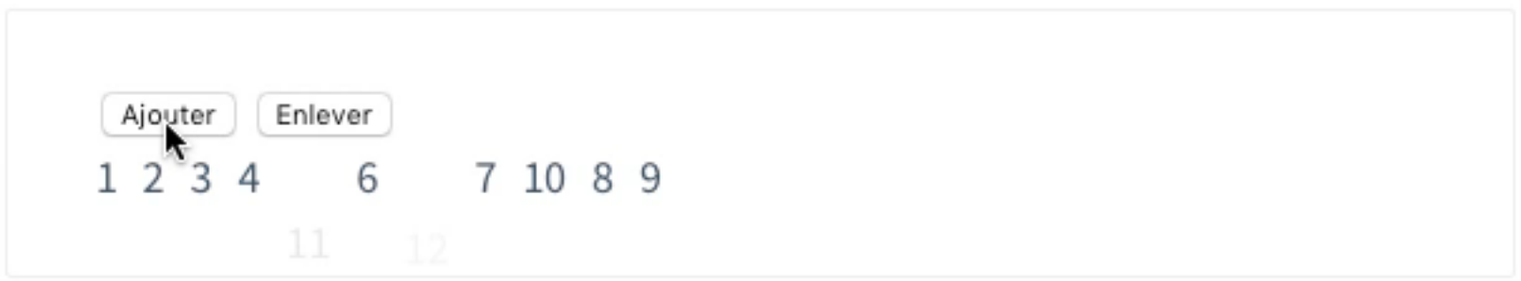
<transition-group>

- <transition> ne permet de gérer la transition que d'un élément dans son contenu
- Si on veut pouvoir traiter les éléments d'une liste avec la même transition, il faut utiliser le composant <transition-group>
- <transition-group> va encapsuler tous les sous-éléments dans un tag réel ""
- Le tag peut être modifié <transition-group tag="div">
- Chaque élément rendu sous le composant doit avoir une propriété key unique

Transitions & Animations 14

<transition-group>

```
<div id="list-demo">
  <button v-on:click="add">Ajouter</button>
  <button v-on:click="remove">Enlever</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>
```




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

Transitions & Animations 15

v-move

- Vue.js détecte les mouvements d’éléments dans une liste

- Chaque élément en mouvement se voit doter d’une classe v-move

Transitions & Animations 16

v-move

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


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