eBook Gratuit

APPRENEZ Vue.js

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#vue.js

Table des matières

À propos 1

Chapitre 1: Démarrer avec Vue.js 2

Remarques 2

Versions 2

Examples 2

"Bonjour le monde!" Programme 2

Exemple simple 2

Modèle HTML 3

JavaScript 3

Bonjour tout le monde dans Vue 2 (la manière JSX) 3

Gestion des entrées utilisateur 4

Chapitre 2: Bus d'événement 5

Introduction 5

Syntaxe 5

Remarques 5

Examples 5

eventBus 5

Chapitre 3: Composants 7

Remarques 7

Examples 7

Composant de portée (non global) 7

HTML 7

JS 7

Quels sont les composants et comment définir les composants? 8

Enregistrement local des composants 11

Inscription en ligne 11

Enregistrement de données dans des composants 11

Événements 12

Chapitre 4: Composants dynamiques 14

Remarques 14

Examples 14
Exemple de composants dynamiques simples 14
Javascript: 14
HTML: 14
Fragment: 14
Navigation de pages avec keep-alive 15
Javascript: 15
HTML: 16
CSS: 16
Fragment: 16
Chapitre 5: Composants personnalisés avec v-model 17
Introduction 17
Remarques 17
Examples 17
v-model sur un composant de compteur 17
Chapitre 6: Crochets de cycle de vie 19
Examples 19
Crochets pour Vue 1.x 19
init 19
created 19
beforeCompile 19
compiled 19
ready 19
attached 19
detached 19
beforeDestroy 19
destroyed 19
Utiliser dans une instance 20
Pièges courants: Accès à DOM depuis le hook `ready ()` 20
Chapitre 7: Des fentes 22
Remarques 22

Examples ... 22
Utiliser des slots individuels ... 22
Que sont les slots? ... 23
Utiliser des slots nommés ... 24
Utiliser des slots dans Vue JSX avec 'babel-plugin-transform-vue-jsx' ... 25
Chapitre 8: Directives sur mesure ... 26
Syntaxe ... 26
Paramètres ... 26
Examples ... 26
Les bases ... 26
Chapitre 9: Événements ... 30
Examples ... 30
Syntaxe des événements ... 30
Quand devrais-je utiliser des événements? ... 30
L'exemple ci-dessus peut être amélioré! ... 32
Comment faire face à la dépréciation de $ dispatch et $ broadcast? (modèle d'événement de ... 33
Chapitre 10: Filtres personnalisés ... 35
Syntaxe ... 35
Paramètres ... 35
Examples ... 35
Filtres à deux voies ... 35
De base ... 36
Chapitre 11: Les accessoires ... 37
Remarques ... 37
camelCase <=> kebab-case ... 37
Examples ... 37
Transmission de données de parent à enfant avec des accessoires ... 37
Accessoires dynamiques ... 42
JS ... 42
HTML ... 43
Résultat ... 43
Passer les accessoires lors de l'utilisation de Vue JSX ... 43

ParentComponent.js 43
ChildComponent.js: 43
Chapitre 12: Les modifications de détection de la matrice 45
Introduction 45
Examples 45
Utilisation de Vue. $ Set 45
Utiliser Array.prototype.splice 45
Pour un tableau imbriqué 46
Tableau d'objets contenant des tableaux 46
Chapitre 13: Liaison de données 47
Examples 47
Texte 47
HTML brut 47
Les attributs 47
Des filtres 47
Chapitre 14: Mixins 49
Examples 49
Mixin Global 49
Stratégies de fusion d'options personnalisées 49
Les bases 50
Option Fusion 50
Chapitre 15: Modèle "webpack" Polyfill 52
Paramètres 52
Remarques 52
Examples 52
Utilisation des fonctions pour polyfill (ex: find) 52
Chapitre 16: Modificateurs 53
Introduction 53
Examples 53
Modificateurs d'événement 53
Modificateurs de clés 53
Modificateurs d'entrée 54

Chapitre 17: Observateurs 55
Examples 55
Comment ça marche 55
Chapitre 18: Plugins 57
Introduction 57
Syntaxe 57
Paramètres 57
Remarques 57
Examples 57
Enregistreur simple 57
Chapitre 19: Propriétés calculées 59
Remarques 59
Données vs propriétés calculées 59
Examples 59
Exemple de base 59
Propriétés calculées contre montre 60
Setters Calculés 60
Utilisation de paramètres calculés pour v-model 61
Chapitre 20: Rendu conditionnel 64
Syntaxe 64
Remarques 64
Examples 64
Vue d'ensemble 64
v-if 64
v-else 64
v-show 64
v-if / v-else 64
v-show 66
Chapitre 21: Rendu de la liste 67
Examples 67
Utilisation de base 67

HTML 67
Scénario 67
Ne rendre que les éléments HTML 67
Liste de compte à rebours 67
Itération sur un objet 68
Chapitre 22: Utiliser "this" dans Vue 69
Introduction 69
Examples 69
FAUX! Utiliser "this" dans un rappel dans une méthode Vue 69
FAUX! Utiliser "this" dans une promesse 69
DROITE! Utilisez une fermeture pour capturer "ceci" 69
DROITE! Utilisez bind 70
DROITE! Utilisez une fonction de flèche 70
FAUX! Utiliser une fonction de flèche pour définir une méthode qui fait référence à "this" 70
DROITE! Définir des méthodes avec la syntaxe de fonction typique 71
Chapitre 23: Vue des composants d'un seul fichier 72
Introduction 72
Examples 72
Exemple de fichier de composant .vue 72
Chapitre 24: VueJS + Redux avec Vua-Redux (meilleure solution) 73
Examples 73
Comment utiliser Vua-Redux 73
Initialiser: 73
Chapitre 25: vue-routeur 76
Introduction 76
Syntaxe 76
Examples 76
Routage de base 76
Chapitre 26: Vuex 77
Introduction 77
Examples 77
Qu'est ce que Vuex? 77

Pourquoi utiliser Vuex? 80
Comment installer Vuex? 82
Notifications de licenciement automatique 82
Crédits 86

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: vue-js

It is an unofficial and free Vue.js ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Vue.js.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec Vue.js

Remarques

`Vue.js` est un framework frontal en pleine expansion pour `JavaScript` , inspiré par `Angular.js` , `Reactive.js` et `Rivets.js` qui offre une conception, une manipulation et une réactivité approfondies de l'interface utilisateur.

Il est décrit comme une structure à `Model-View View-Model MVVM` , `Model-View View-Model` , qui repose sur le concept de *liaison de* données *bidirectionnelle* aux composants et aux vues. Il est incroyablement rapide, dépasse les vitesses des autres frameworks `JS` haut niveau et est très convivial pour faciliter l'intégration et le prototypage.

Versions

Version	Date de sortie
2.4.1	2017-07-13
2.3.4	2017-06-08
2.3.3	2017-05-09
2.2.6	2017-03-26
2.0.0	2016-10-02
1.0.26	2016-06-28
1.0.0	2015-10-26
0.12.0	2015-06-12
0.11.0	2014-11-06

Examples

"Bonjour le monde!" Programme

Pour commencer à utiliser Vue.js , assurez-vous que le fichier de script est inclus dans votre code HTML. Par exemple, ajoutez ce qui suit à votre code HTML.

```
<script src="https://npmcdn.com/vue/dist/vue.js"></script>
```

Exemple simple

Modèle HTML

```
<div id="app">
  {{ message }}
</div>
```

JavaScript

```
new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue.js!'
  }
})
```

Voir une démonstration en direct de cet exemple.

Vous voudrez peut-être aussi consulter l'exemple "Hello World" réalisé par Vue.js.

Bonjour tout le monde dans Vue 2 (la manière JSX)

JSX n'est pas destiné à être interprété par le navigateur. Il faut d'abord le transposer en Javascript standard. Pour utiliser JSX, vous devez installer le plugin pour babel `babel-plugin-transform-vue-JSX`

Exécutez la commande ci-dessous:

```
npm install babel-plugin-syntax-jsx babel-plugin-transform-vue-jsx babel-helper-vue-jsx-merge-props --save-dev
```

et l'ajouter à votre `.babelrc` comme ceci:

```
{
  "presets": ["es2015"],
  "plugins": ["transform-vue-jsx"]
}
```

Exemple de code avec VUE JSX:

```
import Vue from 'vue'
import App from './App.vue'

new Vue({
  el: '#app',
  methods: {
    handleClick () {
```

```
      alert('Hello!')
    }
  },
  render (h) {
    return (
      <div>
        <h1 on-click={this.handleClick}>Hello from JSX</h1>
        <p> Hello World </p>
      </div>
    )
  }
})
```

En utilisant JSX, vous pouvez écrire des structures HTML / XML concises dans le même fichier que vous écrivez du code JavaScript.

Félicitations, vous êtes fait :)

Gestion des entrées utilisateur

VueJS peut également être utilisé pour gérer facilement les entrées utilisateur, et la liaison bidirectionnelle à l'aide de v-model facilite grandement la modification des données.

HTML:

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<div id="app">
    {{message}}
<input v-model="message">
</div>
```

JS:

```
new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue.js!'
  }
})
```

Il est très facile d'effectuer une liaison bidirectionnelle dans VueJS en utilisant la directive `v-model` .

Consultez un exemple en direct ici.

Lire Démarrer avec Vue.js en ligne: https://riptutorial.com/fr/vue-js/topic/1057/demarrer-avec-vue-js

Chapitre 2: Bus d'événement

Introduction

Les bus d'événements sont un moyen utile de communiquer entre des composants qui ne sont pas directement liés, c'est-à-dire sans relation parent-enfant.

C'est juste une instance de `vue` vide, qui peut être utilisée pour `$emit` événements ou écouter `$on` lesdits événements.

Syntaxe

1. export par défaut nouveau Vue ()

Remarques

Utilisez vuex si votre application contient beaucoup de composants nécessitant les données les uns des autres.

Examples

eventBus

```
// setup an event bus, do it in a separate js file
var bus = new Vue()

// imagine a component where you require to pass on a data property
// or a computed property or a method!

Vue.component('card', {
  template: `<div class='card'>
    Name:
      <div class='margin-5'>
          <input v-model='name'>
    </div>
    <div class='margin-5'>
      <button @click='submit'>Save</button>
    </div>
  </div>`,
  data() {
    return {
      name: null
    }
  },
  methods: {
    submit() {
      bus.$emit('name-set', this.name)
    }
  }
})
```

```
// In another component that requires the emitted data.
var data = {
  message: 'Hello Vue.js!'
}

var demo = new Vue({
  el: '#demo',
  data: data,
  created() {
    console.log(bus)
    bus.$on('name-set', (name) => {
      this.message = name
    })
  }
})
```

Lire Bus d'événement en ligne: https://riptutorial.com/fr/vue-js/topic/9498/bus-d-evenement

Chapitre 3: Composants

Remarques

En composant (s):

props est un tableau de littéraux de chaîne ou de références d'objet utilisés pour transmettre des données du composant parent. Il peut également être sous forme d'objet lorsque l'on souhaite avoir un contrôle plus fin, comme la spécification des valeurs par défaut, le type de données accepté, qu'il soit obligatoire ou facultatif.

data doit être une fonction qui renvoie un objet au lieu d'un objet simple. C'est parce que nous avons besoin de chaque instance du composant pour avoir ses propres données à des fins de réutilisation.

événements est un objet contenant des écouteurs pour les événements auxquels le composant peut répondre par un changement de comportement

méthodes objet contenant des fonctions définissant le comportement associé au composant

les propriétés **calculées** sont comme les observateurs ou les observables, chaque fois qu'une dépendance change, les propriétés sont recalculées automatiquement et les modifications sont reflétées instantanément dans DOM si DOM utilise des propriétés calculées

ready est le crochet du cycle de vie d'une instance de Vue

Examples

Composant de portée (non global)

Démo

HTML

```
<script type="x-template" id="form-template">
    <label>{{inputLabel}} :</label>
  <input type="text" v-model="name" />
</script>

<div id="app">
  <h2>{{appName}}</h2>
  <form-component title="This is a form" v-bind:name="userName"></form-component>
</div>
```

JS

```
// Describe the form component
// Note: props is an array of attribute your component can take in entry.
// Note: With props you can pass static data('title') or dynamic data('userName').
// Note: When modifying 'name' property, you won't modify the parent variable, it is only
descendent.
// Note: On a component, 'data' has to be a function that returns the data.
var formComponent = {
  template: '#form-template',
  props: ['title', 'name'],
  data: function() {
    return {
      inputLabel: 'Name'
    }
  }
};

// This vue has a private component, it is only available on its scope.
// Note: It would work the same if the vue was a component.
// Note: You can build a tree of components, but you have to start the root with a 'new
Vue()'.
var vue = new Vue({
  el: '#app',
  data: {
    appName: 'Component Demo',
    userName: 'John Doe'
  },
  components: {
    'form-component': formComponent
  }
});
```

Quels sont les composants et comment définir les composants?

Les composants dans Vue sont comme des widgets. Ils nous permettent d'écrire des éléments personnalisés réutilisables avec le comportement souhaité.

Ce ne sont que des objets pouvant contenir toutes les options que la racine ou toute instance de Vue peut contenir, y compris un modèle HTML à rendre.

Les composants comprennent:

- Balisage HTML: le modèle du composant
- Styles CSS: comment le balisage HTML sera affiché
- Code JavaScript: les données et le comportement

Celles-ci peuvent être écrites dans un fichier séparé ou dans un fichier unique avec l'extension `.vue` . Vous trouverez ci-dessous des exemples montrant les deux manières:

.VUE - en tant que fichier unique pour le composant

```
<style>
    .hello-world-compoment{
        color:#eeeeee;
        background-color:#555555;
    }
</style>
```

```
<template>
    <div class="hello-world-component">
        <p>{{message}}</p>
        <input @keyup.enter="changeName($event)"/>
    </div>
</template>

<script>
    export default{
        props:[ /* to pass any data from the parent here... */ ],
        events:{ /* event listeners go here */},
        ready(){
            this.name= "John";
        },
        data(){
           return{
              name:''
           }
        },
        computed:{
            message(){
                return "Hello from " + this.name;
            }
        },
        methods:{
            // this could be easily achieved by using v-model on the <input> field, but just
to show a method doing it this way.
            changeName(e){
                this.name = e.target.value;
            }
        }
    }
</script>
```

Fichiers séparés

hello-world.js - le fichier JS pour l'objet composant

```
export default{
    template:require('./hello-world.template.html'),
    props:[ /* to pass any data from the parent here... */ ],
    events:{ /* event listeners go here */ },
    ready(){
        this.name="John";
    },
    data(){
        return{
            name:''
        }
    },
    computed:{
        message(){
            return "Hello World! from " + this.name;
        }
    },
    methods:{
        changeName(e){
            let name = e.target.value;
            this.name = name;
```

```
        }
    }
}
```

hello-world.template.html

```
<div class="hello-world-component">
    <p>{{message}}</p>
    <input class="form-control input-sm" @keyup.enter="changeName($event)">
</div>
```

bonjour-world.css

```
 .hello-world-compoment{
    color:#eeeeee;
    background-color:#555555;
}
```

Ces exemples utilisent la syntaxe es2015, de sorte que Babel sera nécessaire pour les compiler en es5 pour les anciens navigateurs.
Babel avec **Browserify + vueify** ou **Webpack + vue-loader** sera nécessaire pour compiler `hello-world.vue` .

Maintenant que nous avons le composant `hello-world` défini, nous devrions l'enregistrer avec Vue.

Ceci peut être fait de deux façons:

S'inscrire en tant que composant global
Dans le fichier `main.js` (point d'entrée de l'application), nous pouvons enregistrer tous les composants globalement avec `Vue.component` :

```
import Vue from 'vue'; // Note that 'vue' in this case is a Node module installed with 'npm
install Vue'
Vue.component('hello-world', require('./hello-world');  // global registeration

new Vue({
    el:'body',

    // Templates can be defined as inline strings, like so:
    template:'<div class="app-container"><hello-world></hello-world></div>'
});
```

Ou l'enregistrer localement dans un composant parent ou un composant racine

```
import Vue from 'vue'; // Note that 'vue' in this case is a Node module installed with 'npm
install Vue'
import HelloWorld from './hello-world.js';

new Vue({
    el:'body',
    template:'<div class="app-container"><hello-world></hello-world></div>",

    components:{HelloWorld}  // local registeration
```

```
});
```

Les composants globaux peuvent être utilisés n'importe où dans l'application Vue.

Les composants locaux ne sont disponibles que pour utilisation dans le composant parent avec lequel ils sont enregistrés.

Composant de fragment
Vous pouvez obtenir une erreur de console vous indiquant que vous ne pouvez pas faire quelque chose car votre *composant* est un *fragment* . Pour résoudre ce type de problème, placez simplement votre template de composant dans une balise unique, comme un `<div>` .

Enregistrement local des composants

Un composant peut être enregistré globalement ou localement (lié à un autre composant spécifique).

```
var Child = Vue.extend({
    // ...
})

var Parent = Vue.extend({
    template: '...',
    components: {
         'my-component': Child
    }
})
```

Le nouveau composant () ne sera disponible que dans la portée (modèle) du composant Parent.

Inscription en ligne

Vous pouvez étendre et enregistrer un composant en une seule étape:

```
Vue.component('custom-component', {
    template: '<div>A custom component!</div>'
})
```

Aussi lorsque le composant est enregistré localement:

```
var Parent = Vue.extend({
    components: {
        'custom-component': {
            template: '<div>A custom component!</div>'
        }
    }
})
```

Enregistrement de données dans des composants

Si vous transmettez un objet à la propriété `data` lors de l'enregistrement d'un composant, toutes

les instances du composant pointeraient sur les mêmes données. Pour résoudre ce problème, nous devons renvoyer des `data` d'une fonction.

```
var CustomComponent = Vue.extend({
    data: function () {
        return { a: 1 }
    }
})
```

Événements

Un des moyens par lesquels les composants peuvent communiquer avec leurs ancêtres / descendants est via des événements de communication personnalisés. Toutes les instances de Vue sont également des émetteurs et implémentent une interface d'événement personnalisée qui facilite la communication dans une arborescence de composants. Nous pouvons utiliser les éléments suivants:

- `$on` : écoute les événements émis par ses ancêtres ou descendants.
- `$broadcast` : `$broadcast` un événement qui se propage vers le bas à tous les descendants.
- `$dispatch` : émet un événement qui se déclenche d'abord sur le composant lui-même et qui se propage vers le haut à tous les ancêtres.
- `$emit` : déclenche un événement sur lui-même.

Par exemple, nous voulons masquer un composant de bouton spécifique dans un composant de formulaire lorsque le formulaire est soumis. Sur l'élément parent:

```
var FormComponent = Vue.extend({
  // ...
  components: {
    ButtonComponent
  },
  methods: {
    onSubmit () {
        this.$broadcast('submit-form')
    }
  }
})
```

Sur l'élément enfant:

```
var FormComponent = Vue.extend({
  // ...
  events: {
    'submit-form': function () {
        console.log('I should be hiding');
    }
  }
})
```

Quelques points à garder en tête:

- Chaque fois qu'un événement trouve un composant qui l'écoute et qui est déclenché, il

cesse de se propager à moins que le rappel de la fonction dans ce composant ne retourne `true` .

- `$dispatch()` déclenche toujours d'abord sur le composant qui l'a émis.
- Nous pouvons transmettre n'importe quel nombre d'arguments au gestionnaire d'événements. Faire `this.$broadcast('submit-form', this.formData, this.formStatus)` nous permet d'accéder à ces arguments comme `'submit-form': function (formData, formStatus) {}`

Lire Composants en ligne: https://riptutorial.com/fr/vue-js/topic/1775/composants

Chapitre 4: Composants dynamiques

Remarques

`<component>` est un élément de composant réservé, à ne pas confondre avec l'instance de composant.

`v-bind` est une directive. Les directives sont préfixées par *v-* pour indiquer qu'elles sont des attributs spéciaux fournis par Vue.

Examples

Exemple de composants dynamiques simples

Basculer dynamiquement entre plusieurs composants à l' aide de l' élément `<component>` et transmettre des données à *v-bind* :

Javascript:

```
new Vue({
  el: '#app',
  data: {
    currentPage: 'home'
  },
  components: {
    home: {
      template: "<p>Home</p>"
    },
    about: {
      template: "<p>About</p>"
    },
    contact: {
      template: "<p>Contact</p>"
    }
  }
})
```

HTML:

```
<div id="app">
   <component v-bind:is="currentPage">
       <!-- component changes when currentPage changes! -->
       <!-- output: Home -->
   </component>
</div>
```

Fragment:

Démo en direct

Navigation de pages avec keep-alive

Parfois, vous voulez garder les composants désactivés en mémoire, pour que cela se produise, vous devez utiliser l'élément `<keep-alive>` :

Javascript:

```
new Vue({
  el: '#app',
  data: {
    currentPage: 'home',
  },
  methods: {
     switchTo: function(page) {
            this.currentPage = page;
     }
  },
  components: {
    home: {
      template: `<div>
                 <h2>Home</h2>
                 <p>{{ homeData }}</p>
                 </div>`,
      data: function() {
         return {
            homeData: 'My about data'
         }
       }
     },
    about: {
      template: `<div>
                 <h2>About</h2>
                 <p>{{ aboutData }}</p>
                 </div>`,
      data: function() {
         return {
            aboutData: 'My about data'
         }
      }
    },
    contact: {
      template: `<div>
             <h2>Contact</h2>
             <form method="POST" @submit.prevent>
             <label>Your Name:</label>
             <input type="text" v-model="contactData.name" >
             <label>You message: </label>
             <textarea v-model="contactData.message"></textarea>
             <button type="submit">Send</button>
             </form>
             </div>`,
       data: function() {
          return {
            contactData: { name:'', message:'' }
         }
      }
```

```
    }
  }
})
```

HTML:

```
<div id="app">
  <div class="navigation">
    <ul>
      <li><a href="#home" @click="switchTo('home')">Home</a></li>
      <li><a href="#about" @click="switchTo('about')">About</a></li>
      <li><a href="#contact" @click="switchTo('contact')">Contact</a></li>
    </ul>
  </div>

  <div class="pages">
    <keep-alive>
      <component :is="currentPage"></component>
    </keep-alive>
  </div>
</div>
```

CSS:

```
.navigation {
  margin: 10px 0;
}

.navigation ul {
  margin: 0;
  padding: 0;
}

.navigation ul li {
  display: inline-block;
  margin-right: 20px;
}

input, label, button {
  display: block
}

input, textarea {
  margin-bottom: 10px;
}
```

Fragment:

Démo en direct

Lire Composants dynamiques en ligne: https://riptutorial.com/fr/vue-js/topic/7702/composants-dynamiques

Chapitre 5: Composants personnalisés avec v-model

Introduction

Souvent, nous devons créer des composants qui effectuent des actions / opérations sur des données et nous en avons besoin dans le composant parent. La plupart du temps, la `vuex` serait une meilleure solution, mais dans les cas où le comportement du composant enfant n'a rien à voir avec l'état de l'application, par exemple: un curseur de plage, un sélecteur de date / heure, un lecteur de fichier

Les magasins individuels pour chaque composant à chaque utilisation deviennent compliqués.

Remarques

Pour avoir `v-model` sur un composant, vous devez remplir deux conditions.

1. Il devrait avoir un accessoire nommé "value"
2. Il doit émettre un événement d' `input` avec la valeur attendue par les composants parents.

```
<component v-model='something'></component>
```

est juste du sucre syntaxique pour

```
<component
    :value="something"
    @input="something = $event.target.value"
>
</component>
```

Examples

v-model sur un composant de compteur

Ici `counter` est un composant enfant accessible par la `demo` qui est un composant parent utilisant `v-model` .

```
// child component
Vue.component('counter', {
  template: `<div><button @click='add'>+1</button>
  <button @click='sub'>-1</button>
  <div>this is inside the child component: {{ result }}</div></div>`,
  data () {
    return {
      result: 0
```

```
    }
  },
  props: ['value'],
  methods: {
    emitResult () {
      this.$emit('input', this.result)
    },
    add () {
      this.result += 1
      this.emitResult()
    },
    sub () {
      this.result -= 1
      this.emitResult()
    }
  }
})
```

Ce composant enfant émettra un `result` chaque fois que les méthodes `sub()` ou `add()` seront appelées.

```
// parent component
new Vue({
  el: '#demo',
  data () {
    return {
      resultFromChild: null
    }
  }
})

// parent template
<div id='demo'>
  <counter v-model='resultFromChild'></counter>
  This is in parent component {{ resultFromChild }}
</div>
```

Comme `v-model` est présent sur le composant enfant, un prop avec la `value` name a été envoyé en même temps, il y a un événement d'entrée sur le `counter` qui à son tour fournira la valeur du composant enfant.

Lire Composants personnalisés avec v-model en ligne: https://riptutorial.com/fr/vue-js/topic/9353/composants-personnalises-avec-v-model

Chapitre 6: Crochets de cycle de vie

Examples

Crochets pour Vue 1.x

- **`init`**

  Appelé de manière synchrone après l'initialisation de l'instance et avant toute observation de données initiale.

- **`created`**

  Appelé de manière synchrone après la création de l'instance. Cela se produit avant l'installation de `$el` , mais après `data observation` , `computed properties` , `watch/event callbacks` et les `methods` ont été configurés.

- **`beforeCompile`**

  Immédiatement avant la compilation de l'instance de Vue.

- **`compiled`**

  Immédiatement après la compilation est terminée. Toutes les `directives` sont liées mais toujours avant que `$el` soit disponible.

- **`ready`**

  Se produit après la compilation *et que* `$el` sont terminés et que l'instance est injectée dans le DOM pour la première fois.

- **`attached`**

  Se produit lorsque `$el` est attaché au DOM par une `directive` ou un appel d'instance `$appendTo()` .

- **`detached`**

  Appelé lorsque `$el` est supprimé / détaché du DOM ou de la méthode d'instance.

- **`beforeDestroy`**

  Immédiatement avant que l'instance de Vue ne soit détruite, mais reste totalement fonctionnelle.

- **`destroyed`**

Appelé après la destruction d'une instance. Toutes les `bindings` et `directives` ont déjà été non liées et les instances enfants ont également été détruites.

Utiliser dans une instance

Étant donné que *tous* les crochets du cycle de vie dans `Vue.js` ne sont que des `functions` , vous pouvez placer l' *un* d'eux directement dans la declaraction instance.

```
//JS
new Vue({

    el: '#example',

    data: {
        ...
    },

    methods: {
        ...
    },

    //LIFECYCLE HOOK HANDLING
    created: function() {
        ...
    },

    ready: function() {
        ...
    }

});
```

Pièges courants: Accès à DOM depuis le hook `ready ()`

Une des utilisations courantes du hook `ready()` est d'accéder au DOM, par exemple pour lancer un plugin Javascript, obtenir les dimensions d'un élément, etc.

Le problème

En raison du mécanisme de mise à jour DOM asynchrone de Vue, il n'est pas garanti que le DOM ait été entièrement mis à jour lorsque le hook `ready()` est appelé. Cela entraîne généralement une erreur car l'élément n'est pas défini.

La solution

Pour cette situation, la méthode d'instance `$nextTick()` peut aider. Cette méthode diffère l'exécution de la fonction de rappel fournie jusqu'à la prochaine coche, ce qui signifie qu'elle est déclenchée lorsque toutes les mises à jour du DOM sont garanties.

Exemple:

```
module.exports {
  ready: function () {
```

```
    $('.cool-input').initiateCoolPlugin() //fails, because element is not in DOM yet.

    this.$nextTick(function() {
      $('.cool-input').initiateCoolPlugin() // this will work because it will be executed
after the DOM update.
    })
  }
}
```

Lire Crochets de cycle de vie en ligne: https://riptutorial.com/fr/vue-js/topic/1852/crochets-de-cycle-de-vie

Chapitre 7: Des fentes

Remarques

Important! Les emplacements après rendu ne garantissent pas l'ordre des positions pour les emplacements. Slot, qui était le premier, peut avoir une position différente après le rendu.

Examples

Utiliser des slots individuels

Les emplacements uniques sont utilisés lorsqu'un composant enfant définit uniquement un `slot` dans son modèle. Le composant de `page` ci-dessus utilise un seul emplacement pour distribuer le contenu.

Un exemple de la `page` du modèle du composant à l' aide d' une seule fente est ci - dessous:

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <slot>
            This will only be displayed if there is no content
            to be distributed.
        </slot>
    </body>
</html>
```

Pour illustrer le fonctionnement de la machine à sous, nous pouvons configurer une page comme suit.

```
<page>
    <p>This content will be displayed within the page component</p>
</page>
```

Le résultat final sera:

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <p>This content will be displayed within the page component</p>
    </body>
</html>
```

Si nous ne mettons rien entre la `page` par balises INSTEAD avaient `<page></page>` nous cédons à la place le résultat suivant car il est contenu par défaut entre les `slot` balises dans la `page` par modèle

des composants.

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        This will only be displayed if there is no content
        to be distributed.
    </body>
</html>
```

Que sont les slots?

Les emplacements offrent un moyen pratique de distribuer du contenu d'un composant parent à un composant enfant. Ce contenu peut être n'importe quoi, du texte, du HTML ou même d'autres composants.

Il peut être utile parfois de considérer les slots comme un moyen d’injecter du contenu directement dans le modèle d’un composant enfant.

Les emplacements sont particulièrement utiles lorsque la composition du composant sous le composant parent n'est pas toujours la même.

Prenons l'exemple suivant où nous avons un composant de `page` . Le contenu de la page peut changer selon que cette page affiche par exemple un article, un article de blog ou un formulaire.

Article

```
<page>
    <article></article>
    <comments></comments>
</page>
```

Blog Post

```
<page>
    <blog-post></blog-post>
    <comments></comments>
</page>
```

Forme

```
<page>
    <form></form>
</page>
```

Notez comment le contenu du composant de `page` peut changer. Si nous n'utilisions pas de slots, cela serait plus difficile car la partie interne du template serait corrigée.

Rappelez-vous: *"Tout dans le modèle parent est compilé dans la portée parent; tout dans le*

modèle enfant est compilé dans la portée enfant."

Utiliser des slots nommés

Les slots nommés fonctionnent de la même manière que les slots simples, mais vous permettent de distribuer du contenu dans différentes régions de votre template de composant enfant.

Prenez le composant `page` de l'exemple précédent, mais modifiez son modèle pour qu'il soit comme suit:

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <aside>
            <slot name="sidebar"></slot>
        </aside>
        <main>
            <slot name="content"></slot>
        </main>
    </body>
</html>
```

Lorsque vous utilisez le composant `page` , nous pouvons maintenant déterminer où le contenu est placé via l'attribut `slot` :

```
<page>
    <p slot="sidebar">This is sidebar content.</p>
    <article slot="content"></article>
</page>
```

La page résultante sera:

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <aside>
            <p>This is sidebar content.</p>
        </aside>
        <main>
            <article></article>
        </main>
    </body>
</html>
```

Si un `slot` est défini sans attribut de `name` tout contenu placé dans des balises de composant ne spécifiant pas d'attribut de `slot` sera placé dans cet emplacement.

Voir l'exemple d' insertion multiple sur les documents officiels de Vue.js.

Utiliser des slots dans Vue JSX avec 'babel-plugin-transform-vue-jsx'

Si vous utilisez VueJS2 et aimez utiliser JSX avec lui. Dans ce cas, pour utiliser le slot, la solution avec exemple est ci-dessous `this.$slots.default` utiliser `this.$slots.default` C'est presque comme `this.props.children` dans React JS.

Component.js:

```
export default {
    render(h) {  //eslint-disable-line
        return (
            <li>
                { this.$slots.default }
            </li>
        );
    }
};
```

ParentComponent.js

```
import Component from './Component';

export default {
    render(h) {  //eslint-disable-line
        return (
            <ul>
                <Component>
                    Hello World
                </Component>
            </ul>
        );
    }
};
```

Lire Des fentes en ligne: https://riptutorial.com/fr/vue-js/topic/4484/des-fentes

Chapitre 8: Directives sur mesure

Syntaxe

- `Vue.directive(id, definition);`
- `Vue.directive(id, update); //when you need only the update function.`

Paramètres

Paramètre	Détails
`id`	String - Identifiant de la directive qui sera utilisé sans le préfixe `v-` . (Ajoutez le préfixe `v-` lors de son utilisation)
`definition`	Object - Un objet de définition peut fournir plusieurs fonctions de hook (toutes facultatives): `bind` , `update` et `unbind`

Examples

Les bases

Outre le jeu de directives par défaut fourni par core, Vue.js vous permet également d'enregistrer des directives personnalisées. Les directives personnalisées fournissent un mécanisme pour mapper les modifications de données à un comportement DOM arbitraire.

Vous pouvez enregistrer une directive personnalisée globale avec la `Vue.directive(id, definition)` , en transmettant une directive id suivie d'un objet de définition. Vous pouvez également enregistrer une directive personnalisée locale en l'incluant dans l'option de `directives` un composant.

Fonctions de crochet

- **bind** : appelé une seule fois, lorsque la directive est liée en premier à l'élément.
- **update** : appelé pour la première fois immédiatement après la `bind` avec la valeur initiale, puis à nouveau chaque fois que la valeur de la liaison change. La nouvelle valeur et la valeur précédente sont fournies comme argument.
- **unbind** : appelé une seule fois, lorsque la directive n'est pas liée à l'élément.

```
Vue.directive('my-directive', {
     bind: function () {
       // do preparation work
       // e.g. add event listeners or expensive stuff
       // that needs to be run only once
     },
     update: function (newValue, oldValue) {
       // do something based on the updated value
```

```
      // this will also be called for the initial value
    },
    unbind: function () {
      // do clean up work
      // e.g. remove event listeners added in bind()
    }
})
```

Une fois inscrit, vous pouvez l'utiliser dans les modèles Vue.js comme ceci (n'oubliez pas d'ajouter le préfixe `v-`):

```
<div v-my-directive="someValue"></div>
```

Lorsque vous n'avez besoin que de la fonction de `update` , vous pouvez transmettre une seule fonction à la place de l'objet de définition:

```
Vue.directive('my-directive', function (value) {
  // this function will be used as update()
})
```

Propriétés d'instance directive

Toutes les fonctions de crochet seront copiés dans l'objet directive réelle, que vous pouvez accéder à l' intérieur de ces fonctions comme `this` contexte. L'objet directive expose certaines propriétés utiles:

- **el** : l'élément auquel la directive est liée.
- **vm** : le contexte ViewModel qui possède cette directive.
- **expression** : expression de la liaison, excluant les arguments et les filtres.
- **arg** : l'argument, si présent.
- **name** : le nom de la directive, sans le préfixe.
- **modifiers** : un objet contenant des modificateurs, le cas échéant.
- **descriptor** : objet contenant le résultat d'analyse de la directive entière.
- **params** : un objet contenant des attributs param. Expliqué ci-dessous.

  Vous devez traiter toutes ces propriétés en lecture seule et ne jamais les modifier. Vous pouvez également associer des propriétés personnalisées à l'objet directive, mais veillez à ne pas remplacer accidentellement des objets internes existants.

Un exemple de directive personnalisée utilisant certaines de ces propriétés:

HTML

```
<div id="demo" v-demo:hello.a.b="msg"></div>
```

JavaScript

```
Vue.directive('demo', {
  bind: function () {
    console.log('demo bound!')
```

```
  },
  update: function (value) {
    this.el.innerHTML =
      'name - '       + this.name + '<br>' +
      'expression - ' + this.expression + '<br>' +
      'argument - '   + this.arg + '<br>' +
      'modifiers - '  + JSON.stringify(this.modifiers) + '<br>' +
      'value - '      + value
  }
})
var demo = new Vue({
  el: '#demo',
  data: {
    msg: 'hello!'
  }
})
```

Résultat

```
name - demo
expression - msg
argument - hello
modifiers - {"b":true,"a":true}
value - hello!
```

Littéral d'objet

Si votre directive nécessite plusieurs valeurs, vous pouvez également transmettre un littéral d'objet JavaScript. Rappelez-vous que les directives peuvent prendre n'importe quelle expression JavaScript valide:

HTML

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

JavaScript

```
Vue.directive('demo', function (value) {
  console.log(value.color) // "white"
  console.log(value.text) // "hello!"
})
```

Modificateur littéral

Lorsqu'une directive est utilisée avec le modificateur literal, sa valeur d'attribut sera interprétée comme une chaîne de caractères et transmise directement à la méthode de `update` . La méthode de `update` sera également appelée une seule fois, car une chaîne simple ne peut pas être réactive.

HTML

```
<div v-demo.literal="foo bar baz">
```

JavaScript

```
Vue.directive('demo', function (value) {
  console.log(value) // "foo bar baz"
})
```

Lire Directives sur mesure en ligne: https://riptutorial.com/fr/vue-js/topic/2368/directives-sur-mesure

Chapitre 9: Événements

Examples

Syntaxe des événements

Pour envoyer un événement: `vm.$emit('new-message');`

Pour attraper un événement: `vm.$on('new-message');`

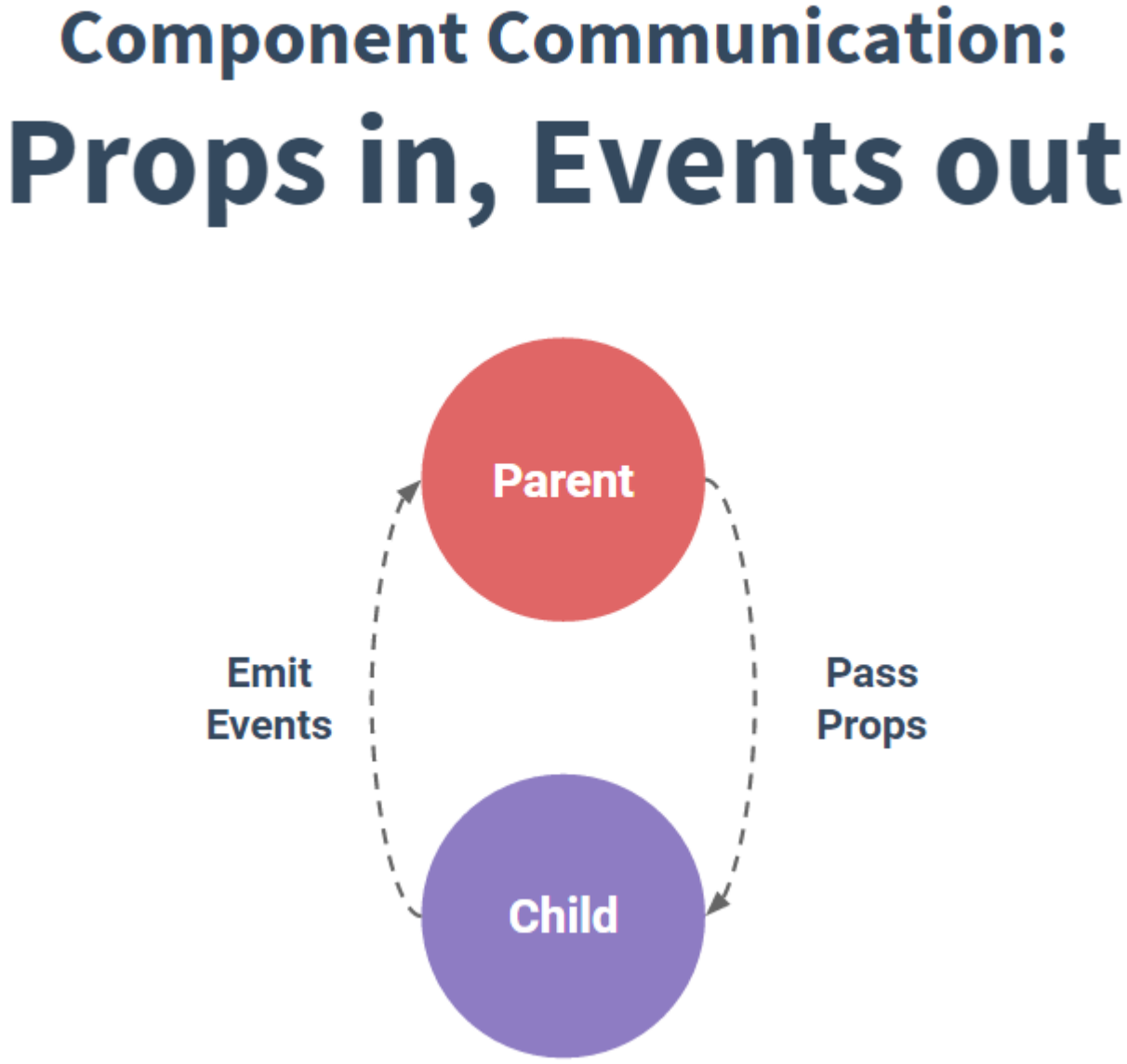
Pour envoyer un événement à tous les composants **vers** le **bas:** `vm.$broadcast('new-message');`

Pour envoyer un événement à tous les composants **jusqu'à:** `vm.$dispatch('new-message');`

Remarque: `$broadcast` et `$dispatch` sont déconseillés dans Vue2. (voir caractéristiques de Vue2)

Quand devrais-je utiliser des événements?

L'image suivante montre comment la communication par composants doit fonctionner. L'image vient du Cadre progressif des diapositives de Evan Vous (développeur de VueJS).



Voici un exemple de fonctionnement:

DEMO

HTML

```
<script type="x-template" id="message-box">
    <input type="text" v-model="msg" @keyup="$emit('new-message', msg)" />
</script>

<message-box :msg="message" @new-message="updateMessage"></message-box>
<div>You typed: {{message}}</div>
```

JS

```
var messageBox = {
  template: '#message-box',
  props: ['msg']
};

new Vue({
  el: 'body',
```

```
  data: {
    message: ''
  },
  methods: {
    updateMessage: function(msg) {
      this.message = msg;
    }
  },
  components: {
    'message-box': messageBox
  }
});
```

L'exemple ci-dessus peut être amélioré!

L'exemple ci-dessus montre le fonctionnement de la communication du composant. Mais dans le cas d'un composant d'entrée personnalisé, pour synchroniser la variable parente avec la valeur saisie, nous utiliserons `v-model` .

DEMO Vue1

DEMO Vue2

Dans Vue1, vous devez utiliser `.sync` sur le prop envoyé au composant `<message-box>` . Cela indique à VueJS de synchroniser la valeur du composant enfant avec celle du parent.

> **N'oubliez pas:** chaque instance de composant a sa propre étendue isolée.

Vue HTML1

```
<script type="x-template" id="message-box">
  <input v-model="value" />
</script>

<div id="app">
  <message-box :value.sync="message"></message-box>
  <div>You typed: {{message}}</div>
</div>
```

Dans Vue2, vous pouvez `$emit` événement spécial `'input'` . L'utilisation de cet événement vous permet de placer un `v-model` virtuel directement sur le composant `<message-box>` . L'exemple ressemblera à ceci:

Vue2 HTML

```
<script type="x-template" id="message-box">
  <input :value="value" @input="$emit('input', $event.target.value)" />
</script>

<div id="app">
  <message-box v-model="message"></message-box>
  <div>You typed: {{message}}</div>
</div>
```

JS Vue 1 & 2

```
var messageBox = {
  template: '#message-box',
  props: ['value']
};

new Vue({
  el: '#app',
  data: {
    message: ''
  },
  components: {
    'message-box': messageBox
  }
});
```

Notez à quelle vitesse l'entrée est mise à jour.

Comment faire face à la dépréciation de $ dispatch et $ broadcast? (modèle d'événement de bus)

Vous avez peut-être réalisé que `$emit` est affecté au composant qui émet l'événement. C'est un problème lorsque vous souhaitez communiquer entre des composants éloignés les uns des autres dans l'arborescence des composants.

Note: Dans Vue1, vous utiliserez `$dispatch` ou `$broadcast` , mais pas dans Vue2. La raison en est que cela ne s'adapte pas bien. Il existe un modèle de `bus` populaire pour gérer cela:

DEMO

HTML

```
<script type="x-template" id="sender">
  <button @click="bus.$emit('new-event')">Click me to send an event !</button>
</script>

<script type="x-template" id="receiver">
  <div>I received {{numberOfEvents}} event{{numberOfEvents == 1 ? '' : 's'}}</div>
</script>

<sender></sender>
<receiver></receiver>
```

JS

```
var bus = new Vue();

var senderComponent = {
  template: '#sender',
  data() {
    return {
      bus: bus
    }
```

```
  }
};

var receiverComponent = {
  template: '#receiver',
  data() {
    return {
      numberOfEvents: 0
    }
  },
  ready() {
    var self = this;

    bus.$on('new-event', function() {
      ++self.numberOfEvents;
    });
  }
};

new Vue({
  el: 'body',
  components: {
    'sender': senderComponent,
    'receiver': receiverComponent
  }
});
```

Vous devez juste comprendre que toute instance de `Vue()` peut `$emit` et intercepter (`$on`) un événement. Nous déclarons simplement un `bus` appel d'instance `Vue` global, puis tout composant avec cette variable peut en émettre et intercepter des événements. Assurez-vous simplement que le composant a accès à la variable de `bus` .

Lire Événements en ligne: https://riptutorial.com/fr/vue-js/topic/5941/evenements

Chapitre 10: Filtres personnalisés

Syntaxe

- `Vue.filter(name, function(value){});` //De base
- `Vue.filter(name, function(value, begin, end){});` // Basic avec les valeurs d'emballage
- `Vue.filter(name, function(value, input){});` //Dynamique
- `Vue.filter(name, { read: function(value){}, write: function(value){} });` // bidirectionnel

Paramètres

Paramètre	Détails
prénom	String - Nom de l'appelable désiré du filtre
valeur	[Rappel] N'importe quelle valeur des données passant dans le filtre
commencer	[Rappel] N'importe quelle valeur à venir avant les données transmises
fin	[Rappel] N'importe quelle valeur à venir après les données transmises
contribution	[Rappel] N'importe - entrée utilisateur liée à l'instance Vue pour des résultats dynamiques

Examples

Filtres à deux voies

Avec un `two-way filter` , nous sommes en mesure d'attribuer une opération de `read` *et d'* `write` pour un `filter` unique qui modifie la valeur des *mêmes* données entre la `view` et le `model` .

```
//JS
Vue.filter('uppercase', {
    //read : model -> view
    read: function(value) {
        return value.toUpperCase();
    },

    //write : view -> model
    write: function(value) {
        return value.toLowerCase();
    }
});

/*
 * Base value of data: 'example string'
 *
 * In the view : 'EXAMPLE STRING'
```

```
 * In the model : 'example string'
 */
```

De base

Les filtres personnalisés dans `Vue.js` peuvent être créés facilement dans un seul appel de fonction à `Vue.filter` .

```
//JS
Vue.filter('reverse', function(value) {
    return value.split('').reverse().join('');
});

//HTML
<span>{{ msg | reverse }}</span> //'This is fun!' => '!nuf si sihT'
```

Il est `./filters` de stocker tous les filtres personnalisés dans des fichiers distincts, par exemple sous `./filters` car il est alors facile de réutiliser votre code dans votre prochaine application. Si vous allez de cette façon, vous devez **remplacer la partie JS** :

```
//JS
Vue.filter('reverse', require('./filters/reverse'));
```

Vous pouvez également définir vos propres wrappers de `begin` et de `end` .

```
//JS
Vue.filter('wrap', function(value, begin, end) {
    return begin + value + end;
});

//HTML
<span>{{ msg | wrap 'The' 'fox' }}</span> //'quick brown' => 'The quick brown fox'
```

Lire Filtres personnalisés en ligne: https://riptutorial.com/fr/vue-js/topic/1878/filtres-personnalises

Chapitre 11: Les accessoires

Remarques

camelCase <=> kebab-case

Lors de la définition des noms de vos `props` , rappelez-vous toujours que les noms d'attributs HTML sont insensibles à la casse. Cela signifie que si vous définissez un `prop` dans un cas camel dans la définition de votre composant ...

```
Vue.component('child', {
    props: ['myProp'],
    ...
});
```

... vous devez l'appeler dans votre composant HTML en tant que ma-prop.

Examples

Transmission de données de parent à enfant avec des accessoires

Dans Vue.js, chaque instance de composant a ***sa propre étendue isolée*** , ce qui signifie que si un composant parent a un composant enfant, le composant enfant a sa propre étendue isolée et le composant parent a sa propre étendue isolée.

Pour toute application de taille moyenne à grande, le respect des conventions de bonnes pratiques évite de nombreux problèmes lors de la phase de développement, puis après la maintenance. Une des choses à suivre est d' ***éviter de référencer ou de muter des données parent directement à partir du composant enfant*** . Alors, comment pouvons-nous référencer les données parent à partir d'un composant enfant?

Toutes les données parentales requises dans un composant enfant doivent être transmises à l'enfant en tant que `props` du parent.

Cas d'utilisation : Supposons que nous ayons une base de données utilisateur avec deux `users` et `addresses` tables avec les champs suivants:
Table d' `users`

prénom	téléphone	email
John Mclane	(1) 234 5678 9012	john@dirhard.com
James Bond	(44) 777 0007 0077	bond@mi6.com

Table d' `addresses`

bloc	rue	ville
Tours Nakatomi	Broadway	New York
Maison Mi6	Buckingham Road	Londres

et nous voulons avoir trois composants pour afficher les informations utilisateur correspondantes n'importe où dans notre application

user-component.js

```
export default{
    template:`<div class="user-component">
                <label for="name" class="form-control">Name: </label>
                <input class="form-control input-sm" name="name" v-model="name">
                <contact-details :phone="phone" :email="email"></contact-details>
              </div>`,
    data(){
        return{
            name:'',
            phone:'',
            email:''
        }
    },
}
```

contact-details.js

```
import Address from './address';
export default{
    template:`<div class="contact-details-component>
                <h4>Contact Details:</h4>
                <label for="phone" class="form-control">Phone: </label>
                <input class="form-control input-sm" name="phone" v-model="phone">
                <label for="email" class="form-control">Email: </label>
                <input class="form-control input-sm" name="email" v-model="email">

                <h4>Address:</h4>
                <address :address-type="addressType"></address>
                //see camelCase vs kebab-case explanation below
            </div>`,
    props:['phone', 'email'],
    data:(){
        return:{
            addressType:'Office'
        }
    },
    components:{Address}
}
```

address.js

```
export default{
    template:`<div class="address-component">
                <h6>{{addressType}}</h6>
                <label for="block" class="form-control">Block: </label>
```

```
                <input class="form-control input-sm" name="block" v-model="block">
                <label for="street" class="form-control">Street: </label>
                <input class="form-control input-sm" name="street" v-model="street">
                <label for="city" class="form-control">City: </label>
                <input class="form-control input-sm" name="city" v-model="city">
            </div>`,
    props:{
        addressType:{
            required:true,
            type:String,
            default:'Office'
        },
    data(){
        return{
            block:'',
            street:'',
            city:''
        }
    }
}
```

main.js

```
import Vue from 'vue';

Vue.component('user-component', require'./user-component');
Vue.component('contact-details', require'./contact-details');

new Vue({
    el:'body'
});
```

index.html

```
...
<body>
    <user-component></user-component>
        ...
</body>
```

Nous affichons les données de `phone` et de `email` , qui sont les propriétés du `user-component` dans `contact-details` qui ne contiennent pas de données de téléphone ou de courrier électronique.

Transmettre des données comme accessoires

Donc, dans le `user-component.js` de la propriété ***template*** , où nous incluons le composant `<contact-details>` , nous transmettons le ***téléphone*** et les données de ***messagerie*** de `<user-component>` (composant parent) à `<contact-details>` (composant enfant) en le liant dynamiquement aux **accessoires** - `:phone="phone"` et `:email="email` identique à `v-bind:phone="phone"` et `v-bind:email="email"`

Props - Reliure dynamique

Comme nous lions dynamiquement les accessoires, tout changement de ***téléphone*** ou de ***courrier électronique*** dans le composant parent, à savoir `<user-component>` sera immédiatement

répercuté dans le composant enfant, c. `<contact-details>` d. `<contact-details>` .

Props - comme littéraux

Cependant, si nous avions passé les valeurs de ***téléphone*** et de ***courrier électronique*** sous la forme de valeurs littérales de type chaîne de caractères telles que `phone="(44) 777 0007 0077" email="bond@mi6.com"` cela ne refléterait aucune modification de données composant.

Reliure à sens unique

Par défaut, la direction des modifications est de haut en bas, c'est-à-dire que toute modification des propriétés liées dynamiquement dans le composant parent se propagera au composant enfant, mais toute modification des valeurs de prop dans un composant enfant ne se propagera pas au parent.

Par exemple: si à l' intérieur des `<contact-details>` nous changeons l'e - mail de `bond@mi6.com` à `jamesbond@mi6.com` , la propriété des données de téléphone données parent à savoir dans `<user-component>` contiendra encore une valeur de `bond@mi6.com` .

Cependant, si nous changeons la valeur de l'email de `bond@mi6.com` à `jamesbond@mi6.co` dans le composant parent (`<user-component>` dans notre cas d'utilisation), alors la valeur de l'email dans le composant enfant (`<contact-details>` dans notre cas d'utilisation) changera automatiquement pour `jamesbond@mi6.com` - le changement de parent est instantanément transmis à l'enfant.

Reliure bidirectionnelle

Si nous voulons une liaison bidirectionnelle, nous devons explicitement spécifier une liaison bidirectionnelle comme `:email.sync="email"` au lieu de `:email="email"` . Maintenant, si nous modifions la valeur de prop dans le composant enfant, la modification sera également répercutée dans le composant parent.

Dans une application de taille moyenne à grande, il sera très difficile de détecter l'état parent à partir de l'état enfant et de le suivre, en particulier lors du débogage. - **Soyez prudent** .

Il n'y aura pas d'option .sync disponible dans Vue.js 2.0. **La liaison bidirectionnelle pour les accessoires est obsolète dans Vue.js 2.0** .

Liaison ponctuelle

Il est également possible de définir **une** liaison explicite **unique** comme `:email.once="email` , il est plus ou moins similaire à passer un littéral, car toute modification ultérieure de la valeur de la propriété parent ne se propagera pas à l'enfant.

CAVEAT
Lorsque **Object ou Array** est passé en prop, ils sont **TOUJOURS PASSÉS PAR REFERENCE** , ce qui signifie quel que soit le type de liaison explicitement défini `:email.sync="email"` ou `:email="email"` ou `:email.once="email"` , Si email est un objet ou un tableau dans le parent, quel que soit le type de liaison, toute modification de la valeur de prop dans le composant enfant affectera également la valeur du parent.

Les accessoires comme tableau

Dans le fichier `contact-details.js` , nous avons défini des `props:['phone', 'email']` comme un tableau, ce qui est bien si nous ne voulons pas un contrôle fin avec des accessoires.

Les accessoires comme objet

Si nous voulons un contrôle plus fin des accessoires, comme

- si nous voulons définir quel type de valeurs sont acceptables comme prop
- quelle devrait être une valeur par défaut pour le prop
- si une valeur DOIT (obligatoire) être passée pour le prop ou est-elle facultative?

alors nous devons utiliser la notation d'objet pour définir les accessoires, comme nous l'avons fait dans `address.js` .

Si nous créons des composants réutilisables pouvant être utilisés par d'autres développeurs de l'équipe, il est recommandé de définir des objets en tant qu'objets afin que toute personne utilisant le composant ait une idée claire de ce que devrait être le type de données et si c'est obligatoire ou facultatif.

Il est également appelé ***validation des accessoires*** . Le **type** peut être l'un des constructeurs natifs suivants:

- Chaîne
- Nombre
- Booléen
- Tableau
- Objet
- Fonction
- ou un constructeur personnalisé

Quelques exemples de validation du prop depuis http://vuejs.org/guide/components.html#Props

```
Vue.component('example', {
   props: {
       // basic type check (`null` means accept any type)
       propA: Number,
       // multiple possible types (1.0.21+)
       propM: [String, Number],
       // a required string
       propB: {
         type: String,
         required: true
       },
       // a number with default value
       propC: {
          type: Number,
          default: 100
       },
       // object/array defaults should be returned from a
       // factory function
       propD: {
```

```
          type: Object,
          default: function () {
             return { msg: 'hello' }
          }
       },
       // indicate this prop expects a two-way binding. will
       // raise a warning if binding type does not match.
       propE: {
          twoWay: true
       },
       // custom validator function
       propF: {
          validator: function (value) {
             return value > 10
          }
       },
       // coerce function (new in 1.0.12)
       // cast the value before setting it on the component
       propG: {
          coerce: function (val) {
            return val + '' // cast the value to string
          }
       },
       propH: {
          coerce: function (val) {
            return JSON.parse(val) // cast the value to Object
          }
       }
    }
});
```

camelCase vs cas de kebab

Les attributs HTML ne sont pas sensibles à la casse, ce qui signifie qu'ils ne peuvent pas différencier `addresstype` et `addressType` . Par conséquent, lorsque vous utilisez des noms de propriétés camelCase en tant qu'attributs, vous devez utiliser leurs équivalents kebab: `addressType` doit être écrit en tant que `address-type` dans l'attribut HTML.

Accessoires dynamiques

Tout comme vous pouvez lier des données d'une vue au modèle, vous pouvez également lier des éléments à l'aide de la même directive v-bind pour transmettre des informations des composants parent à enfant.

JS

```
new Vue({
    el: '#example',
    data: {
        msg: 'hello world'
    }
});

Vue.component('child', {
    props: ['myMessage'],
```

```
    template: '<span>{{ myMessage }}</span>
});
```

HTML

```
<div id="example">
    <input v-model="msg" />
    <child v-bind:my-message="msg"></child>
    <!-- Shorthand ... <child :my-message="msg"></child> -->
</div>
```

Résultat

```
hello world
```

Passer les accessoires lors de l'utilisation de Vue JSX

Nous avons un composant parent: Importer un composant enfant dans ce composant, nous passerons des accessoires via un attribut. Ici, l'attribut est "src" et nous passons aussi le "src".

ParentComponent.js

```
import ChildComponent from './ChildComponent';
export default {
    render(h, {props}) {
        const src = 'https://cdn-images-1.medium.com/max/800/1*AxRXW2j8qmGJixIYg7n6uw.jpeg';
        return (
           <ChildComponent src={src} />
        );
    }
};
```

Et un composant enfant, où il faut passer des accessoires. Nous devons spécifier les accessoires que nous passons.

ChildComponent.js:

```
export default {
    props: ['src'],
    render(h, {props}) {
        return (
            <a href = {props.src} download = "myimage" >
                Click this link
            </a>
        );
    }
};
```

Lire Les accessoires en ligne: https://riptutorial.com/fr/vue-js/topic/3080/les-accessoires

Chapitre 12: Les modifications de détection de la matrice

Introduction

Lorsque vous essayez de définir une valeur d'un élément à un index particulier d'un tableau initialisé dans l'option de données, la vue ne peut pas détecter la modification et ne déclenche pas de mise à jour de l'état. Afin de surmonter cet avertissement, vous devez soit utiliser Vue. $ Set ou utiliser la méthode Array.prototype.splice.

Examples

Utilisation de Vue. $ Set

Dans votre méthode ou tout crochet de cycle de vie qui modifie l'élément de tableau à un index particulier

```
new Vue({
    el: '#app',
    data:{
        myArr : ['apple', 'orange', 'banana', 'grapes']
    },
    methods:{
        changeArrayItem: function(){
            //this will not work
            //myArr[2] = 'strawberry';

            //Vue.$set(array, index, newValue)
            this.$set(this.myArr, 2, 'strawberry');
        }
    }
})
```

Voici le lien vers le violon

Utiliser Array.prototype.splice

Vous pouvez effectuer le même changement au lieu d'utiliser `Vue.$set` en utilisant le `splice()` du prototype Array

```
new Vue({
    el: '#app',
    data:{
        myArr : ['apple', 'orange', 'banana', 'grapes']
    },
    methods:{
        changeArrayItem: function(){
            //this will not work
            //myArr[2] = 'strawberry';
```

```
            //Array.splice(index, 1, newValue)
            this.myArr.splice(2, 1, 'strawberry');
        }
    }
})
```

Pour un tableau imbriqué

Si vous avez un tableau imbriqué, vous pouvez procéder comme suit

```
new Vue({
    el: '#app',
    data:{
        myArr : [
            ['apple', 'banana'],
            ['grapes', 'orange']
        ]
    },
    methods:{
        changeArrayItem: function(){
            this.$set(this.myArr[1], 1, 'strawberry');
        }
    }
})
```

Voici le lien vers le jsfiddle

Tableau d'objets contenant des tableaux

```
new Vue({
    el: '#app',
    data:{
        myArr : [
            {
                name: 'object-1',
                nestedArr: ['apple', 'banana']
            },
            {
                name: 'object-2',
                nestedArr: ['grapes', 'orange']
            }
        ]
    },
    methods:{
        changeArrayItem: function(){
            this.$set(this.myArr[1].nestedArr, 1, 'strawberry');
        }
    }
})
```

Voici le lien vers le violon

Lire Les modifications de détection de la matrice en ligne: https://riptutorial.com/fr/vue-js/topic/10679/les-modifications-de-detection-de-la-matrice

Chapitre 13: Liaison de données

Examples

Texte

La forme la plus élémentaire de la liaison de données est l'interpolation de texte utilisant la syntaxe «Moustache» (accolades doubles):

```
<span>Message: {{ msg }}</span>
```

La balise moustache sera remplacée par la valeur de la propriété `msg` sur l'objet de données correspondant. Il sera également mis à jour chaque fois que la propriété `msg` l'objet de données change.

Vous pouvez également effectuer des interpolations ponctuelles qui ne sont pas mises à jour lors du changement de données:

```
<span>This will never change: {{* msg }}</span>
```

HTML brut

La double moustache interprète les données en texte brut, pas en HTML. Afin de générer du vrai HTML, vous devrez utiliser des moustaches triples:

```
<div>{{{ raw_html }}}</div>
```

Le contenu est inséré en tant que HTML simple - les liaisons de données sont ignorées. Si vous devez réutiliser des éléments de modèle, vous devez utiliser des partiels.

Les attributs

Les moustaches peuvent également être utilisées dans les attributs HTML:

```
<div id="item-{{ id }}"></div>
```

Notez que les interpolations d'attribut sont interdites dans les directives Vue.js et les attributs spéciaux. Ne vous inquiétez pas, Vue.js émettra des avertissements pour vous lorsque des moustaches sont utilisées à des endroits inappropriés.

Des filtres

Vue.js vous permet d'ajouter des «filtres» facultatifs à la fin d'une expression, désignés par le symbole «pipe»:

```
{{ message | capitalize }}
```

Ici, nous «canalisons» la valeur de l'expression du `message` via le filtre intégré en `capitalize` , qui est en fait une simple fonction JavaScript qui renvoie la valeur en majuscule. Vue.js fournit un certain nombre de filtres intégrés, et nous parlerons de la manière d'écrire vos propres filtres ultérieurement.

Notez que la syntaxe de tube ne fait pas partie de la syntaxe JavaScript. Par conséquent, vous ne pouvez pas mélanger les filtres dans les expressions. vous ne pouvez les ajouter qu'à la fin d'une expression.

Les filtres peuvent être chaînés:

```
{{ message | filterA | filterB }}
```

Les filtres peuvent également prendre des arguments:

```
{{ message | filterA 'arg1' arg2 }}
```

La fonction de filtre reçoit toujours la valeur de l'expression comme premier argument. Les arguments cités sont interprétés comme une chaîne simple, tandis que les arguments non cités seront évalués comme des expressions. Ici, la chaîne `'arg1'` caractères `'arg1'` sera transmise dans le filtre en tant que second argument, et la valeur de l'expression `arg2` sera évaluée et transmise en tant que troisième argument.

Lire Liaison de données en ligne: https://riptutorial.com/fr/vue-js/topic/1213/liaison-de-donnees

Chapitre 14: Mixins

Examples

Mixin Global

Vous pouvez également appliquer un mixin globalement. Faites attention! Une fois que vous appliquez un mixin globalement, cela affecte **toutes les** instances de Vue créées ultérieurement. Utilisé correctement, il peut être utilisé pour injecter une logique de traitement pour les options personnalisées:

```
// inject a handler for `myOption` custom option
Vue.mixin({
  created: function () {
    var myOption = this.$options.myOption
    if (myOption) {
      console.log(myOption)
    }
  }
})

new Vue({
  myOption: 'hello!'
})
// -> "hello!"
```

> Utilisez les mixins globaux avec parcimonie et avec précaution, car cela affecte chaque instance de Vue créée, y compris les composants tiers. Dans la plupart des cas, vous ne devez l'utiliser que pour la gestion des options personnalisées, comme le montre l'exemple ci-dessus.

Stratégies de fusion d'options personnalisées

Lorsque les options personnalisées sont fusionnées, elles utilisent la stratégie par défaut, qui remplace simplement la valeur existante. Si vous souhaitez qu'une option personnalisée soit fusionnée à l'aide d'une logique personnalisée, vous devez associer une fonction à `Vue.config.optionMergeStrategies` :

```
Vue.config.optionMergeStrategies.myOption = function (toVal, fromVal) {
  // return mergedVal
}
```

Pour la plupart des options basées sur des objets, vous pouvez simplement utiliser la même stratégie utilisée par les `methods` :

```
var strategies = Vue.config.optionMergeStrategies
strategies.myOption = strategies.methods
```

Les bases

Les mixins constituent un moyen flexible de distribuer des fonctionnalités réutilisables pour les composants Vue. Un objet mixin peut contenir des options de composant. Lorsqu'un composant utilise un mixin, toutes les options du mixin seront mélangées dans les options du composant.

```
// define a mixin object
var myMixin = {
  created: function () {
    this.hello()
  },
  methods: {
    hello: function () {
      console.log('hello from mixin!')
    }
  }
}

// define a component that uses this mixin
var Component = Vue.extend({
  mixins: [myMixin]
})

var component = new Component() // -> "hello from mixin!"
```

Option Fusion

Lorsqu'un mixin et le composant lui-même contiennent des options qui se chevauchent, ils seront «fusionnés» à l'aide de stratégies appropriées. Par exemple, les fonctions de hook avec le même nom sont fusionnées dans un tableau pour qu'elles soient toutes appelées. De plus, les hooks mixin seront appelés **avant** les propres crochets du composant:

```
var mixin = {
  created: function () {
    console.log('mixin hook called')
  }
}

new Vue({
  mixins: [mixin],
  created: function () {
    console.log('component hook called')
  }
})

// -> "mixin hook called"
// -> "component hook called"
```

Les options qui attendent des valeurs d'objet, par exemple des `methods` , des `components` et des `directives` , seront fusionnées dans le même objet. Les options du composant auront la priorité lorsqu'il y a des clés en conflit dans ces objets:

```
var mixin = {
  methods: {
```

```
    foo: function () {
      console.log('foo')
    },
    conflicting: function () {
      console.log('from mixin')
    }
  }
}

var vm = new Vue({
  mixins: [mixin],
  methods: {
    bar: function () {
      console.log('bar')
    },
    conflicting: function () {
      console.log('from self')
    }
  }
})

vm.foo() // -> "foo"
vm.bar() // -> "bar"
vm.conflicting() // -> "from self"
```

Notez que les mêmes stratégies de fusion sont utilisées dans `Vue.extend()` .

Lire Mixins en ligne: https://riptutorial.com/fr/vue-js/topic/2562/mixins

Chapitre 15: Modèle "webpack" Polyfill

Paramètres

Fichiers ou paquets	Commande ou configuration à modifier
babel-polyfill	`npm i -save babel-polyfill`
karma.conf.js	`files: ['../../node_modules/babel-polyfill/dist/polyfill.js','./index.js'],`
webpack.base.conf.js	`app: ['babel-polyfill', './src/main.js']`

Remarques

Les configurations décrites ci-dessus, l'exemple utilisant une fonction non standardisée, fonctionneront sur "Internet Explorer" et le `npm test` passera.

Examples

Utilisation des fonctions pour polyfill (ex: find)

```
<template>
  <div class="hello">
    <p>{{ filtered() }}</p>
  </div>
</template>

<script>
export default {
  name: 'hello',
  data () {
    return {
      list: ['toto', 'titi', 'tata', 'tete']
    }
  },
  methods: {
    filtered () {
      return this.list.find((el) => el === 'tata')
    }
  }
}
</script>
```

Lire Modèle "webpack" Polyfill en ligne: https://riptutorial.com/fr/vue-js/topic/9174/modele--webpack--polyfill

Chapitre 16: Modificateurs

Introduction

Il existe des opérations fréquemment utilisées, telles que `event.preventDefault()` ou `event.stopPropagation()` dans les gestionnaires d'événements. Bien que nous puissions faire cela facilement à l'intérieur des méthodes, il serait préférable que les méthodes concernent uniquement la logique des données plutôt que d'avoir à gérer les détails des événements DOM.

Examples

Modificateurs d'événement

Vue fournit des modificateurs d'événement pour `v-on` en appelant des postfixes de directive désignés par un point.

- `.stop`
- `.prevent`
- `.capture`
- `.self`
- `.once`

Pour des exemples:

```
<!-- the click event's propagation will be stopped -->
<a v-on:click.stop="doThis"></a>

<!-- the submit event will no longer reload the page -->
<form v-on:submit.prevent="onSubmit"></form>

<!-- use capture mode when adding the event listener -->
<div v-on:click.capture="doThis">...</div>

<!-- only trigger handler if event.target is the element itself -->
<!-- i.e. not from a child element -->
<div v-on:click.self="doThat">...</div>
```

Modificateurs de clés

Lors de l'écoute d'événements de clavier, nous avons souvent besoin de rechercher des codes de clé communs. La mémorisation de tous les keyCodes est un problème, et Vue fournit des alias pour les clés les plus utilisées:

- `.enter`
- `.tab`
- `.delete` (capture les touches “Delete” et “Backspace”)
- `.esc`
- `.space`
- `.up`

- `.down`
- `.left`
- `.right`

Pour des exemples:

```
<input v-on:keyup.enter="submit">
```

Modificateurs d'entrée

- `.trim`

Si vous souhaitez que les entrées utilisateur soient automatiquement ajustées, vous pouvez ajouter le modificateur `trim` à vos entrées gérées par `v-model` :

```
<input v-model.trim="msg">
```

- `.number`

Si vous souhaitez que l'entrée utilisateur soit automatiquement convertie en un nombre, vous pouvez procéder comme suit:

```
<input v-model.number="age" type="number">
```

- `.lazy`

Généralement, `v-model` synchronise l'entrée avec les données après chaque événement d'entrée, mais vous pouvez ajouter le modificateur `lazy` à la place de la synchronisation après les événements de modification:

```
<input v-model.lazy="msg" >
```

Lire Modificateurs en ligne: https://riptutorial.com/fr/vue-js/topic/8612/modificateurs

Chapitre 17: Observateurs

Examples

Comment ça marche

Vous pouvez regarder la propriété de données de toute instance de Vue. Lorsque vous regardez une propriété, vous déclenchez une méthode sur le changement:

```
export default {
    data () {
        return {
            watched: 'Hello World'
        }
    },
    watch: {
        'watched' () {
            console.log('The watched property has changed')
        }
    }
}
```

Vous pouvez récupérer l'ancienne valeur et la nouvelle:

```
export default {
    data () {
        return {
            watched: 'Hello World'
        }
    },
    watch: {
        'watched' (value, oldValue) {
            console.log(oldValue) // Hello World
            console.log(value) // ByeBye World
        }
    },
    mounted () {
        this.watched = 'ByeBye World'
    }
}
```

Si vous devez regarder des propriétés imbriquées sur un objet, vous devrez utiliser la propriété `deep` :

```
export default {
    data () {
        return {
            someObject: {
                message: 'Hello World'
            }
        }
    },
    watch: {
```

```
        'someObject': {
            deep: true,
            handler (value, oldValue) {
                console.log('Something changed in someObject')
            }
        }
    }
}
```

Quand les données sont-elles mises à jour?

Si vous devez déclencher l'observateur avant d'apporter de nouvelles modifications à un objet, vous devez utiliser la méthode `nextTick()` :

```
export default {
    data() {
        return {
            foo: 'bar',
            message: 'from data'
        }
    },
    methods: {
        action () {
            this.foo = 'changed'
            // If you juste this.message = 'from method' here, the watcher is executed after.
            this.$nextTick(() => {
                this.message = 'from method'
            })
        }
    },
    watch: {
        foo () {
            this.message = 'from watcher'
        }
    }
}
```

Lire Observateurs en ligne: https://riptutorial.com/fr/vue-js/topic/7988/observateurs

Chapitre 18: Plugins

Introduction

Les plugins Vue ajoutent des fonctionnalités globales telles que des méthodes globales, des directives, des transitions, des filtres, des méthodes d'instance, des objets et injectent des options de composants en utilisant des mixins.

Syntaxe

- MyPlugin.install = function (Vue, options) {}

Paramètres

prénom	La description
Vue	Vue constructeur, injecté par Vue
options	Options supplémentaires si nécessaire

Remarques

Dans la plupart des cas, vous devrez explicitement demander à Vue d'utiliser un plugin

```
// calls `MyPlugin.install(Vue)`
Vue.use(MyPlugin)
```

Pour passer des options

```
Vue.use(MyPlugin, { someOption: true })
```

Examples

Enregistreur simple

```
//myLogger.js
export default {

  install(Vue, options) {
     function log(type, title, text) {
       console.log(`[${type}] ${title} - ${text}`);
     }

     Vue.prototype.$log = {
```

```
      error(title, text) { log('danger', title, text) },
      success(title, text) { log('success', title, text) },
      log
    }
  }
}
```

Avant que votre instance principale de Vue dise d'enregistrer votre plugin

```
//main.js
import Logger from './path/to/myLogger';

Vue.use(Logger);

var vm = new Vue({
  el: '#app',
  template: '<App/>',
  components: { App }
})
```

Maintenant, vous pouvez appeler `this.$log` sur n'importe quel composant enfant

```
//myComponent.vue
export default {
  data() {
    return {};
  },
  methods: {
    Save() {
      this.$log.success('Transaction saved!');
    }
  }
}
```

Lire Plugins en ligne: https://riptutorial.com/fr/vue-js/topic/8726/plugins

Chapitre 19: Propriétés calculées

Remarques

Données vs propriétés calculées

La principale différence de cas d'utilisation pour les `data` et `computed` propriétés `computed` d'une instance de `Vue` dépend de l'état potentiel ou de la probabilité de modification des données. Lorsque vous décidez de la catégorie d'un objet, ces questions peuvent vous aider:

- Est-ce une valeur constante? (**données**)
- Est-ce que cela a la possibilité de changer? (**calculé** ou **données**)
- Est-ce que sa valeur dépend de la valeur des autres données? (**calculé**)
- A-t-il besoin de données ou de calculs supplémentaires avant d'être utilisé? (**calculé**)
- La valeur ne changera-t-elle que dans certaines circonstances? (**données**)

Examples

Exemple de base

Modèle

```
<div id="example">
  a={{ a }}, b={{ b }}
</div>
```

JavaScript

```
var vm = new Vue({
  el: '#example',
  data: {
    a: 1
  },
  computed: {
    // a computed getter
    b: function () {
      // `this` points to the vm instance
      return this.a + 1
    }
  }
})
```

Résultat

```
a=1, b=2
```

Ici, nous avons déclaré une propriété calculée `b` . La fonction que nous avons fournie sera utilisée

comme fonction getter pour la propriété `vm.b` :

```
console.log(vm.b) // -> 2
vm.a = 2
console.log(vm.b) // -> 3
```

La valeur de `vm.b` dépend toujours de la valeur de `vm.a`

Vous pouvez lier des données aux propriétés calculées dans des modèles, tout comme une propriété normale. Vue est conscient que `vm.b` dépend de `vm.a` , donc il mettra à jour toutes les liaisons qui dépendent de `vm.b` lorsque `vm.a` change.

Propriétés calculées contre montre

modèle

```
<div id="demo">{{fullName}}</div>
```

exemple de montre

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar',
    fullName: 'Foo Bar'
  }
})

vm.$watch('firstName', function (val) {
  this.fullName = val + ' ' + this.lastName
})

vm.$watch('lastName', function (val) {
  this.fullName = this.firstName + ' ' + val
})
```

Exemple calculé

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar'
  },
  computed: {
    fullName: function () {
      return this.firstName + ' ' + this.lastName
    }
  }
})
```

Setters Calculés

Les propriétés calculées seront automatiquement recalculées à chaque modification des données sur lesquelles le calcul est effectué. Cependant, si vous devez modifier manuellement une propriété calculée, Vue vous permet de créer une méthode de réglage pour ce faire:

Template (de l'exemple de base ci-dessus):

```
<div id="example">
  a={{ a }}, b={{ b }}
</div>
```

Javascript:

```
var vm = new Vue({
  el: '#example',
  data: {
    a: 1
  },
  computed: {
    b: {
      // getter
      get: function () {
        return this.a + 1
      },
      // setter
      set: function (newValue) {
        this.a = newValue - 1
      }
    }
  }
```

Vous pouvez maintenant appeler le getter ou le setter:

```
console.log(vm.b)       // -> 2
vm.b = 4                // (setter)
console.log(vm.b)       // -> 4
console.log(vm.a)       // -> 3
```

`vm.b = 4` invoquera le setter et définira this.a à 3; par extension, vm.b sera évalué à 4.

Utilisation de paramètres calculés pour v-model

Vous pourriez avoir besoin d'un `v-model` sur une propriété calculée. Normalement, le modèle virtuel ne met pas à jour la valeur de la propriété calculée.

Le gabarit:

```
<div id="demo">
  <div class='inline-block card'>
    <div :class='{onlineMarker: true, online: status, offline: !status}'></div>
    <p class='user-state'>User is {{ (status) ? 'online' : 'offline' }}</p>
  </div>

  <div class='margin-5'>
```

```
    <input type='checkbox' v-model='status'>Toggle status (This will show you as offline to
others)
  </div>
</div>
```

Coiffant:

```
#demo {
  font-family: Helvetica;
  font-size: 12px;
}
.inline-block > * {
  display: inline-block;
}
.card {
  background: #ddd;
  padding:2px 10px;
  border-radius: 3px;
}
.onlineMarker {
  width: 10px;
  height: 10px;
  border-radius: 50%;
  transition: all 0.5s ease-out;
}

.online {
  background-color: #3C3;
}

.offline {
  background-color: #aaa;
}

.user-state {
  text-transform: uppercase;
  letter-spacing: 1px;
}

.margin-5 {
  margin: 5px;
}
```

Le composant:

```
var demo = new Vue({
    el: '#demo',
    data: {
        statusProxy: null
    },
    computed: {
        status: {
            get () {
                return (this.statusProxy === null) ? true : this.statusProxy
            }
        }
    }
})
```

violon Ici, vous verrez que cliquer sur le bouton radio n'a aucune utilité, votre statut est toujours en ligne.

```
var demo = new Vue({
    el: '#demo',
    data: {
        statusProxy: null
    },
    computed: {
        status: {
            get () {
                return (this.statusProxy === null) ? true : this.statusProxy
            },
            set (val) {
                this.statusProxy = val
            }
        }
    }
})
```

violon Et maintenant, vous pouvez voir que la bascule se produit lorsque la case est cochée / décochée.

Lire Propriétés calculées en ligne: https://riptutorial.com/fr/vue-js/topic/2371/proprietes-calculees

Chapitre 20: Rendu conditionnel

Syntaxe

- `<element v-if="condition"></element>` // v-if
- `<element v-if="condition"></element><element v-else="condition"></element>` // v-if | v-else
- `<template v-if="condition">...</template>` // modèle avec v-if
- `<element v-show="condition"></element>` // v-show

Remarques

Il est très important de se rappeler la différence entre `v-if` et `v-show` . Bien que leurs utilisations soient presque identiques, un élément lié à `v-if` *ne sera rendu que dans le DOM* lorsque sa condition est `true` pour la **première fois** . Lorsque vous utilisez la directive `v-show` , *tous les éléments **sont** rendus dans le DOM* mais sont masqués en utilisant le style d' `display` si la condition est `false` !

Examples

Vue d'ensemble

Dans Vue.js, le rendu conditionnel est obtenu en utilisant un ensemble de directives sur les éléments du modèle.

`v-if`

L'élément s'affiche normalement lorsque la condition est `true` . Lorsque la condition est `false` , seule *une* compilation *partielle* se produit et l'élément n'est pas rendu dans le DOM tant que la condition n'est plus `true` .

`v-else`

N'accepte pas une condition, mais rend l'élément si la condition `v-if` l'élément précédent est `false` . Ne peut être utilisé qu'après un élément avec la directive `v-if` .

`v-show`

Se comporte de la même manière que `v-if` , cependant, l'élément sera *toujours* rendu dans le DOM, même si la condition est `false` . Si la condition est `false` , cette directive définira simplement le style d' `display` l'élément sur `none` .

v-if / v-else

En supposant que nous ayons une instance de `Vue.js` définie comme suit:

```
var vm = new Vue({
    el: '#example',
    data: {
        a: true,
        b: false
    }
});
```

Vous pouvez rendre conditionnellement n'importe quel élément HTML en incluant la directive v-if; l'élément qui contient v-if ne sera rendu que si la condition est true:

```
<!-- will render 'The condition is true' into the DOM -->
<div id="example">
    <h1 v-if="a">The condition is true</h1>
</div>
```

L'élément `<h1>` sera rendu dans ce cas, car la variable 'a' est vraie. v-if peut être utilisé avec n'importe quelle expression, propriété calculée ou fonction évaluée à un booléen:

```
<div v-if="0 === 1">                  false; won't render</div>
<div v-if="typeof(5) === 'number'">   true; will render</div>
```

Vous pouvez utiliser un élément de `template` pour regrouper plusieurs éléments pour une seule condition:

```
<!-- in this case, nothing will be rendered except for the containing 'div' -->
<div id="example">
    <template v-if="b">
        <h1>Heading</h1>
        <p>Paragraph 1</p>
        <p>Paragraph 2</p>
    </template>
</div>
```

Lorsque vous utilisez `v-if` , vous avez également la possibilité d'intégrer une condition de compteur avec la directive `v-else` . Le contenu contenu dans l'élément ne sera affiché que si la condition de la v-if précédente était fausse. Notez que cela signifie qu'un élément avec v-else doit apparaître immédiatement après un élément avec v-if.

```
<!-- will render only 'ELSE' -->
<div id="example">
    <h1 v-if="b">IF</h1>
    <h1 v-else="a">ELSE</h1>
</div>
```

Tout comme avec v-if, avec v-else, vous pouvez regrouper plusieurs éléments HTML dans un `<template>` :

```
<div v-if="'a' === 'b'"> This will never be rendered. </div>
<template v-else>
    <ul>
      <li> You can also use templates with v-else. </li>
```

```
      <li> All of the content within the template </li>
      <li> will be rendered. </li>
    </ul>
</template>
```

v-show

L'utilisation de la directive `v-show` est presque identique à celle de `v-if` . Les seules différences sont que `v-show` *ne prend pas en* charge la syntaxe `<template>` , et qu'il n'y a pas de condition "alternative".

```
var vm = new Vue({
    el: '#example',
    data: {
        a: true
    }
});
```

L'utilisation de base est la suivante ...

```
<!-- will render 'Condition met' -->
<div id="example">
    <h1 v-show="a">Condition met</h1>
</div>
```

Bien que `v-show` ne prenne pas en charge la directive `v-else` pour définir des conditions "alternatives", cela peut être accompli en annulant la précédente ...

```
<!-- will render 'This is shown' -->
<div id="example">
    <h1 v-show="!a">This is hidden</h1>
    <h1 v-show="a">This is shown</h1>
</div>
```

Lire Rendu conditionnel en ligne: https://riptutorial.com/fr/vue-js/topic/3465/rendu-conditionnel

Chapitre 21: Rendu de la liste

Examples

Utilisation de base

Une liste peut être rendue en utilisant la directive `v-for` . La syntaxe requiert que vous spécifiiez le tableau source sur lequel effectuer une itération, et un alias qui sera utilisé pour référencer chaque élément de l'itération. Dans l'exemple suivant, nous utilisons des `items` tant que tableau source et `item` tant qu'alias pour chaque élément.

HTML

```
<div id="app">
  <h1>My List</h1>
  <table>
    <tr v-for="item in items">
      <td>{{item}}</td>
    </tr>
  </table>
</div>
```

Scénario

```
new Vue({
  el: '#app',
  data: {
    items: ['item 1', 'item 2', 'item 3']
  }
})
```

Vous pouvez voir une démo qui fonctionne ici .

Ne rendre que les éléments HTML

Dans cet exemple, affichera cinq balises `<li>`

```
<ul id="render-sample">
  <li v-for="n in 5">
    Hello Loop
  </li>
</ul>
```

Liste de compte à rebours

```
<ul>
  <li v-for="n in 10">{{11 - n}} pigs are tanning at the beach. One got fried, and
```

```
</ul>
```

https://jsfiddle.net/gurghet/3jeyka22/

Itération sur un objet

`v-for` peut être utilisé pour itérer sur un objet keys (et des valeurs):

HTML:

```
<div v-for="(value, key) in object">
  {{ key }} : {{ value }}
</div>
```

Scénario:

```
new Vue({
  el: '#repeat-object',
  data: {
    object: {
      FirstName: 'John',
      LastName: 'Doe',
      Age: 30
    }
  }
})
```

Lire Rendu de la liste en ligne: https://riptutorial.com/fr/vue-js/topic/1972/rendu-de-la-liste

Chapitre 22: Utiliser "this" dans Vue

Introduction

L' une des erreurs les plus courantes que nous trouvons dans le code Vue sur StackOverflow est l'utilisation abusive de `this` . Les erreurs les plus courantes tombent généralement dans deux domaines, en utilisant `this` dans callbacks de promesses ou d' autres fonctions asynchrones et en utilisant des fonctions de direction pour définir les méthodes, les propriétés de calcul, etc.

Examples

FAUX! Utiliser "this" dans un rappel dans une méthode Vue.

```
new Vue({
  el:"#app",
  data:{
    foo: "bar"
  },
  methods:{
    doSomethingAsynchronous(){
      setTimeout(function(){
        // This is wrong! Inside this function,
        // "this" refers to the window object.
        this.foo = "baz";
      }, 1000);
    }
  }
})
```

FAUX! Utiliser "this" dans une promesse.

```
new Vue({
  el:"#star-wars-people",
  data:{
    people: null
  },
  mounted: function(){
    $.getJSON("http://swapi.co/api/people/", function(data){
      // Again, this is wrong! "this", here, refers to the window.
      this.people = data.results;
    })
  }
})
```

DROITE! Utilisez une fermeture pour capturer "ceci"

Vous pouvez saisir la bonne `this` aide d' une fermeture .

```
new Vue({
  el:"#star-wars-people",
```

```
  data:{
    people: null
  },
  mounted: function(){
    // Before executing the web service call, save this to a local variable
    var self = this;
    $.getJSON("http://swapi.co/api/people/", function(data){
      // Inside this call back, because of the closure, self will
      // be accessible and refers to the Vue object.
      self.people = data.results;
    })
  }
})
```

DROITE! Utilisez bind.

Vous pouvez lier la fonction de rappel.

```
new Vue({
  el:"#star-wars-people",
  data:{
    people: null
  },
  mounted:function(){
    $.getJSON("http://swapi.co/api/people/", function(data){
      this.people = data.results;
    }.bind(this));
  }
})
```

DROITE! Utilisez une fonction de flèche.

```
new Vue({
  el:"#star-wars-people",
  data:{
    people: null
  },
  mounted: function(){
    $.getJSON("http://swapi.co/api/people/", data => this.people = data.results);
  }
})
```

Mise en garde! Fonctions fléchées sont une syntaxe introduite en Ecmascript 2015. Il n'a pas encore été pris en charge mais *tous* les navigateurs modernes, donc ne l' utiliser que si vous ciblez un navigateur que vous *connaissez* supporte, ou si vous compilez votre javascript vers le bas à la syntaxe ES5 en utilisant quelque chose comme babel .

FAUX! Utiliser une fonction de flèche pour définir une méthode qui fait référence à "this"

```
new Vue({
  el:"#app",
  data:{
```

```
    foo: "bar"
  },
  methods:{
    // This is wrong! Arrow functions capture "this" lexically
    // and "this" will refer to the window.
    doSomething: () => this.foo = "baz"
  }
})
```

DROITE! Définir des méthodes avec la syntaxe de fonction typique

```
new Vue({
  el:"#app",
  data:{
    foo: "bar"
  },
  methods:{
    doSomething: function(){
      this.foo = "baz"
    }
  }
})
```

Alternativement, si vous utilisez un compilateur javascript ou un navigateur qui supporte Ecmascript 2015

```
new Vue({
  el:"#app",
  data:{
    foo: "bar"
  },
  methods:{
    doSomething(){
      this.foo = "baz"
    }
  }
})
```

Lire Utiliser "this" dans Vue en ligne: https://riptutorial.com/fr/vue-js/topic/9350/utiliser--this--dans-vue

Chapitre 23: Vue des composants d'un seul fichier

Introduction

Décrire comment créer des composants de fichier unique dans un fichier .vue.

Spécialement les décisions de conception qui peuvent être prises.

Examples

Exemple de fichier de composant .vue

```
<template>
  <div class="nice">Component {{title}}</div>
</template>

<script>
export default {
    data() {
        return {
            title: "awesome!"
        };
    }
}
</script>

<style>
.nice {
    background-color: red;
    font-size: 48px;
}
</style>
```

Lire Vue des composants d'un seul fichier en ligne: https://riptutorial.com/fr/vue-js/topic/10118/vue-des-composants-d-un-seul-fichier

Chapitre 24: VueJS + Redux avec Vua-Redux (meilleure solution)

Examples

Comment utiliser Vua-Redux

Installation de Vua Redux à partir de NPM:

Installer à travers:

```
npm i vua-redux --save
```

Initialiser:

==========

// main.js

```
import Vue from 'vue';
import { reduxStorePlugin } from 'vua-redux';
import AppStore from './AppStore';
import App from './Component/App';

// install vua-redux
Vue.use(reduxStorePlugin);

new Vue({
    store: AppStore,
    render(h) {
        return <App />
    }
});
```

// AppStore.js

```
import { createStore } from 'redux';

const initialState = {
  todos: []
};

const reducer = (state = initialState, action) => {
  switch(action.type){
    case 'ADD_TODO':
      return {
        ...state,
        todos: [...state.todos, action.data.todo]
      }
```

```
    default:
      return state;
    }
}

const AppStore = createStore(reducer);

export default AppStore;
```

Utilisez dans votre composant:

// composants / App.js

```
import { connect } from 'vua-redux';

const App = {
    props: ['some-prop', 'another-prop'],

    /**
     * everything you do with vue component props
     * you can do inside collect key
     */
    collect: {
        todos: {
            type: Array,
        },
        addTodo: {
            type: Function,
        },
    },

    methods: {
        handleAddTodo() {
            const todo = this.$refs.input.value;
            this.addTodo(todo);
        }
    },

    render(h) {
        return <div>
            <ul>
                {this.todos.map(todo => <li>{todo}</li>)}
            </ul>

            <div>
                <input type="text" ref="input" />
                <button on-click={this.handleAddTodo}>add todo</button>
            </div>
        </div>
    }
};

function mapStateAsProps(state) {
    return {
        todos: state.todos
    };
}

function mapActionsAsProps(dispatch) {
    return {
```

```
        addTodo(todo) {
            dispatch({
                type: 'ADD_TODO',
                data: { todo }
            })
        }
    }
}

export default connect(mapStateAsProps, mapActionsAsProps)(App);
```

Lire VueJS + Redux avec Vua-Redux (meilleure solution) en ligne: https://riptutorial.com/fr/vue-js/topic/7396/vuejs-plus-redux-avec-vua-redux--meilleure-solution-

Chapitre 25: vue-routeur

Introduction

vue-router est la bibliothèque de routage officiellement prise en charge pour vue.js.

Syntaxe

- `<router-link to="/path">Link Text</router-link> <!-- Creates a link to the route that matches the path -->`
- `<router-view></router-view> <!-- Outlet for the currently matched route. It's component will be rendered here. -->`

Examples

Routage de base

Le moyen le plus simple de se familiariser avec vue-router est d'utiliser la version fournie via CDN.

HTML:

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>

<div id="router-example">
    <router-link to="/foo">Link to Foo route</router-link>
    <router-view></router-view>
</div>
```

JavaScript (ES2015):

```
const Foo = { template: <div>This is the component for the Foo route</div> }

const router = new VueRouter({
    routes: [
       { path: '/foo', component: Foo}
    ]
})

const routerExample = new Vue({
    router
}).$mount('#router-example')
```

Lire vue-routeur en ligne: https://riptutorial.com/fr/vue-js/topic/9654/vue-routeur

Chapitre 26: Vuex

Introduction

Vuex est un modèle de gestion d'état + bibliothèque pour les applications Vue.js. Il sert de magasin centralisé pour tous les composants d'une application, avec des règles garantissant que l'état ne peut être modifié que de manière prévisible. Il s'intègre également à l'extension officielle des outils de développement de Vue pour fournir des fonctionnalités avancées telles que le débogage de voyage temporel sans configuration et l'exportation / importation d'images instantanées.

Examples

Qu'est ce que Vuex?

Vuex est un plugin officiel pour Vue.js qui offre une banque de données centralisée à utiliser dans votre application. Il est fortement influencé par l'architecture de l'application Flux qui comporte un flux de données unidirectionnel qui simplifie la conception et le raisonnement des applications.

Dans une application Vuex, le magasin de données contient tous **les états des applications partagées** . Cet état est modifié par des **mutations** qui sont effectuées en réponse à une **action** invoquant un événement de mutation via le **répartiteur** .

Un exemple du flux de données dans une application Vuex est présenté dans le diagramme ci-dessous.

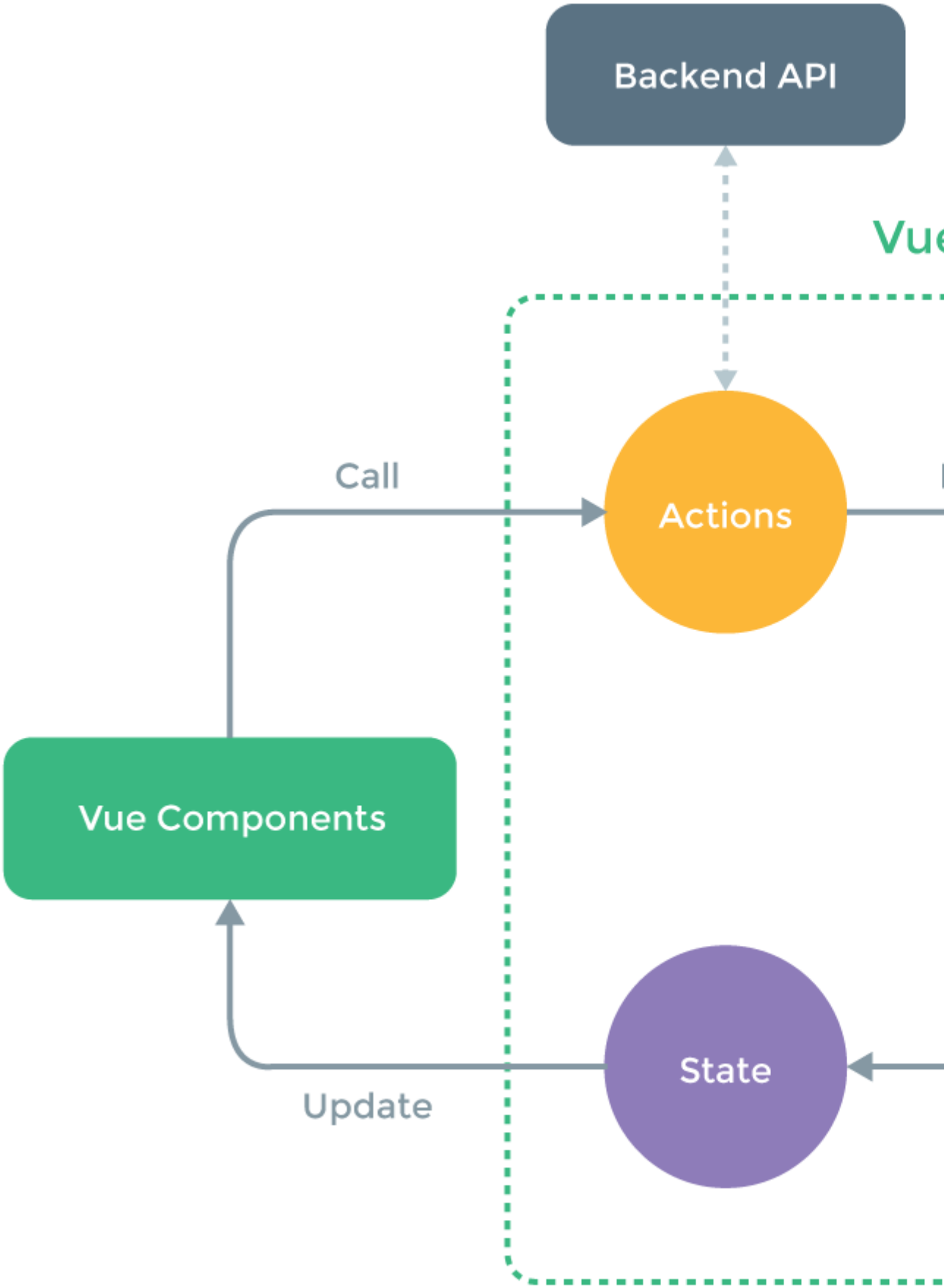

Backend API
Vue
Call
Actions
Vue Components
State
Update

Diagramme utilisé sous licence MIT , à l'origine du dépôt officiel Vuex GitHub .

Les composants individuels de l'application Vue.js peuvent accéder à l'objet store pour récupérer des données via des **getters** , qui sont des fonctions pures renvoyant une copie en lecture seule des données souhaitées.

Les composants peuvent avoir des **actions** qui sont des fonctions qui modifient la copie des données du composant, puis utilisent le **répartiteur** pour envoyer un événement de mutation. Cet événement est ensuite géré par le magasin de données qui met à jour l'état si nécessaire.

Les modifications sont alors automatiquement répercutées dans l'application, car tous les composants sont liés de manière réactive au magasin via leurs getters.

Un exemple illustrant l'utilisation de vuex dans un projet de vue.

```
const state = {
    lastClickTime: null
}

const mutations = {
    updateLastClickTime: (state, payload) => {
      state.lastClickTime = payload
  }
}

const getters = {
  getLastClickTime: state => {
    return new Date(state.lastClickTime)
  }
}

const actions = {
    syncUpdateTime: ({ commit }, payload) => {
    commit("updateLastClickTime", payload)
  },
  asyncUpdateTime: ({ commit }, payload) => {
    setTimeout(() => {
      commit("updateLastClickTime", payload)
    }, Math.random() * 5000)
  }
}

const store = new Vuex.Store({
  state,
  getters,
  mutations,
  actions
})

const { mapActions, mapGetters } = Vuex;

// Vue
const vm = new Vue({
    el: '#container',
  store,
  computed: {
      ...mapGetters([
```

```
        'getLastClickTime'
    ])
  },
  methods: {
      ...mapActions([
        'syncUpdateTime',
      'asyncUpdateTime'
    ]),
    updateTimeSyncTest () {
        this.syncUpdateTime(Date.now())
    },
    updateTimeAsyncTest () {
        this.asyncUpdateTime(Date.now())
    }
  }
})
```

Et le modèle HTML pour le même:

```
<div id="container">
  <p>{{ getLastClickTime || "No time selected yet" }}</p>
  <button @click="updateTimeSyncTest">Sync Action test</button>
  <button @click="updateTimeAsyncTest">Async Action test</button>
</div>
```

1. Ici, l' **état** contient la propriété **lastClickTime** initialisée en tant que null. Cette configuration des valeurs par défaut est importante pour garder les propriétés **réactives** . **Les propriétés non mentionnées dans l'état** seront disponibles mais les modifications apportées par la suite **ne seraient pas accessibles** en utilisant des getters.

2. Le getter utilisé fournit une propriété calculée qui sera mise à jour chaque fois qu'une mutation met à jour la valeur de la propriété state.

3. **Seules les mutations** sont autorisées à modifier l'état et ses propriétés, cela dit, il le fait de manière **synchrone uniquement** .

4. Une Action peut être utilisée en cas de mises à jour asynchrones, où l’appel d’API (ici simulé par setTimeout) peut être effectué dans l’action, et après avoir obtenu la réponse à laquelle une mutation peut être validée, pour passer à l’état. .

Pourquoi utiliser Vuex?

Lors de la création d'applications volumineuses telles que les applications à page unique (SPA), qui consistent généralement en de nombreux composants réutilisables, elles peuvent rapidement devenir difficiles à créer et à gérer. Le partage de données et d’états entre ces composants peut également s’effondrer rapidement et devenir difficile à déboguer et à maintenir.

En utilisant un magasin de données d'application centralisé, l'intégralité de l'état de l'application peut être représentée en un seul endroit, ce qui rend l'organisation plus organisée. Grâce à un flux de données unidirectionnel, à des mutations et à un accès aux données des composants uniquement pour les données requises, il devient beaucoup plus facile de raisonner sur le rôle du composant et sur son impact sur l'état de l'application.

Les composants de VueJS sont des entités distinctes et ils ne peuvent pas partager facilement des données entre eux. Pour partager des données sans vuex nous devons `emit` l' événement avec des données, puis écouter et attraper cet événement avec `on` .

composant 1

```
this.$emit('eventWithDataObject', dataObject)
```

composant 2

```
this.$on('eventWithDataObject', function (dataObject) {
    console.log(dataObject)
})
```

Avec vuex installé, nous pouvons simplement accéder à ses données à partir de n'importe quel composant sans avoir à écouter les événements.

```
this.$store.state.myData
```

Nous pouvons également modifier les données de manière synchrone avec les *mutateurs* , utiliser des *actions* asynchrones et obtenir des données avec des fonctions de *lecture* .

Les fonctions de lecture peuvent fonctionner comme des fonctions calculées globales. Nous pouvons y accéder à partir de composants:

```
this.$store.getters.myGetter
```

Les actions sont des méthodes globales. Nous pouvons les expédier à partir de composants:

```
this.$store.dispatch('myAction', myDataObject)
```

Et les mutations sont le seul moyen de changer les données en vuex. Nous pouvons commettre des modifications:

```
this.$store.commit('myMutation', myDataObject)
```

Le code de Vuex ressemblerait à ceci

```
state: {
  myData: {
    key: 'val'
  }
},
getters: {
  myGetter: state => {
     return state.myData.key.length
  }
},
actions: {
  myAction ({ commit }, myDataObject) {
```

```
    setTimeout(() => {
      commit('myMutation', myDataObject)
    }, 2000)
  }
},
mutations: {
  myMutation (state, myDataObject) {
    state.myData = myDataObject
  }
}
```

Comment installer Vuex?

La plupart du temps, vous utiliserez Vuex dans des applications basées sur des composants de plus grande taille, où vous utiliserez probablement un bundler de modules tel que Webpack ou Browserify en conjonction avec Vueify si vous utilisez des fichiers uniques.

Dans ce cas, le moyen le plus simple d'obtenir Vuex est de NPM. Exécutez la commande ci-dessous pour installer Vuex et l'enregistrer dans les dépendances de votre application.

```
 npm install --save vuex
```

Assurez-vous de charger le lien Vuex avec votre configuration Vue en plaçant la ligne suivante après votre instruction `require('vue')` .

```
Vue.use(require('vuex'))
```

Vuex est également disponible sur CDN; vous pouvez récupérer la dernière version de cdnjs ici .

Notifications de licenciement automatique

Cet exemple enregistrera dynamiquement un module vuex pour stocker des notifications personnalisées pouvant être automatiquement rejetées.

notifications.js

résoudre vuex store et définir des constantes

```
//Vuex store previously configured on other side
import _store from 'path/to/store';

//Notification default duration in milliseconds
const defaultDuration = 8000;

//Valid mutation names
const NOTIFICATION_ADDED = 'NOTIFICATION_ADDED';
const NOTIFICATION_DISMISSED = 'NOTIFICATION_DISMISSED';
```

définir l'état initial de notre module

```
const state = {
```

```
  Notifications: []
}
```

définir nos modules getters

```
const getters = {
  //All notifications, we are returning only the raw notification objects
  Notifications: state => state.Notifications.map(n => n.Raw)
}
```

définir notre module Actions

```
const actions = {
  //On actions we receive a context object which exposes the
  //same set of methods/properties on the store instance
  //{commit} is a shorthand for context.commit, this is an
  //ES2015 feature called argument destructuring
  Add({ commit }, notification) {
    //Get notification duration or use default duration
    let duration = notification.duration || defaultDuration

    //Create a timeout to dismiss notification
    var timeOut = setTimeout(function () {
      //On timeout mutate state to dismiss notification
      commit(NOTIFICATION_DISMISSED, notification);
    }, duration);

    //Mutate state to add new notification, we create a new object
    //for save original raw notification object and timeout reference
    commit(NOTIFICATION_ADDED, {
      Raw: notification,
      TimeOut: timeOut
    })
  },
  //Here we are using context object directly
  Dismiss(context, notification) {
    //Just pass payload
    context.commit(NOTIFICATION_DISMISSED, notification);
  }
}
```

définir nos mutations de module

```
const mutations = {
  //On mutations we receive current state and a payload
  [NOTIFICATION_ADDED](state, notification) {
    state.Notifications.push(notification);
  },
  //remember, current state and payload
  [NOTIFICATION_DISMISSED](state, rawNotification) {
    var i = state.Notifications.map(n => n.Raw).indexOf(rawNotification);
    if (i == -1) {
      return;
    }

    clearTimeout(state.Notifications[i].TimeOut);
    state.Notifications.splice(i, 1);
```

```
  }
}
```

Enregistrez notre module avec un état, des getters, des actions et une mutation définis

```
_store.registerModule('notifications', {
  state,
  getters,
  actions,
  mutations
});
```

Usage

composanteA.vue

Ce composant affiche toutes les notifications en tant qu'alertes bootstrap dans le coin supérieur droit de l'écran, permet également de supprimer manuellement chaque notification.

```
<template>
<transition-group name="notification-list" tag="div" class="top-right">
  <div v-for="alert in alerts" v-bind:key="alert" class="notification alert alert-dismissible"
v-bind:class="'alert-'+alert.type">
    <button v-on:click="dismiss(alert)" type="button" class="close" aria-label="Close"><span
aria-hidden="true">&times;</span></button>
    <div>
      <div>
        <strong>{{alert.title}}</strong>
      </div>
      <div>
        {{alert.text}}
      </div>
    </div>
  </div>
</transition-group>
</template>

<script>
export default {
  name: 'arc-notifications',
  computed: {
    alerts() {
      //Get all notifications from store
      return this.$store.getters.Notifications;
    }
  },
  methods: {
    //Manually dismiss a notification
    dismiss(alert) {
      this.$store.dispatch('Dismiss', alert);
    }
  }
}
</script>
<style lang="scss" scoped>
$margin: 15px;

.top-right {
```

```
    top: $margin;
    right: $margin;
    left: auto;
    width: 300px;
    //height: 600px;
    position: absolute;
    opacity: 0.95;
    z-index: 100;
    display: flex;
    flex-wrap: wrap;
    //background-color: red;
}
.notification {
    transition: all 0.8s;
    display: flex;
    width: 100%;
    position: relative;
    margin-bottom: 10px;
    .close {
        position: absolute;
        right: 10px;
        top: 5px;
    }

    > div {
        position: relative;
        display: inline;
    }
}
.notification:last-child {
    margin-bottom: 0;
}
.notification-list-enter,
.notification-list-leave-active {
    opacity: 0;
    transform: translateX(-90px);
}
.notification-list-leave-active {
    position: absolute;
}
</style>
```

Extrait de code pour ajouter une notification dans tout autre composant

```
//payload could be anything, this example content matches with componentA.vue
this.$store.dispatch('Add', {
  title = 'Hello',
  text = 'World',
  type = 'info',
  duration = 15000
});
```

Lire Vuex en ligne: https://riptutorial.com/fr/vue-js/topic/3430/vuex

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec Vue.js	Community, Erick Petrucelli, ironcladgeek, J. Bruni, James, Lambda Ninja, m_callens, MotKohn, rap-2-h, Ru Chern Chong, Sankalp Singha, Shog9, Shuvo Habib, user1012181, user6939352, Yerko Palma
2	Bus d'événement	Amresh Venugopal
3	Composants	Donkarnash, Elfayer, Hector Lorenzo, Jeff, m_callens, phaberest, RedRiderX, user6939352
4	Composants dynamiques	Med, Ru Chern Chong
5	Composants personnalisés avec v-model	Amresh Venugopal
6	Crochets de cycle de vie	Linus Borg, m_callens, PatrickSteele, xtreak
7	Des fentes	Daniel Waghorn, Elfayer, Shuvo Habib, Slava
8	Directives sur mesure	Mat J, Ogie Sado
9	Événements	Elfayer
10	Filtres personnalisés	Finrod, M U, m_callens
11	Les accessoires	asemahle, Donkarnash, FlatLander, m_callens, rap-2-h, Shuvo Habib
12	Les modifications de détection de la matrice	Vamsi Krishna
13	Liaison de données	gurghet, Jilson Thomas
14	Mixins	Ogie Sado
15	Modèle "webpack" Polyfill	Stefano Nepa

16	Modificateurs	sept08
17	Observateurs	El_Matella
18	Plugins	AldoRomo88
19	Propriétés calculées	Amresh Venugopal, cl3m, jaredsk, m_callens, Theo, Yerko Palma
20	Rendu conditionnel	jaredsk, m_callens, Nirazul, user6939352
21	Rendu de la liste	chuanxd, gurghet, Mahmoud, Theo
22	Utiliser "this" dans Vue	Bert
23	Vue des composants d'un seul fichier	jordiburgos, Ru Chern Chong
24	VueJS + Redux avec Vua-Redux (meilleure solution)	Aniko Litvanyi, FlatLander, Shuvo Habib, Stefano Nepa
25	vue-routeur	AJ Gregory
26	Vuex	AldoRomo88, Amresh Venugopal, Daniel Waghorn, Matej Vrzala M4, Ru Chern Chong