VueJS : Cours

David Mahéo pour les IMR1

Année scolaire 2021-2022

Contents

1 MVVM

MVVM (Modèle Vue, Vue-Modèle) est un design pattern, souvent utilisé par les bibliothèques JavaScript modernes (VueJS, React, ...)qui a été introduit par Microsoft.
Le but étant de séparer vue et données. Le lien entre les deux est fait par des "bindings".

Model Contient les données

View Affichage (le site web)

ViewModel Le lien entre les 2 précédents

2 Installer VueJS

IDE : WebStorm ou VSC avec le plugin Vetur

2.1 Standalone

Listing 1: librairie

```
<script src="https://unpkg.com/vue@2"></script>
```

Listing 2: HTML

```
<div id="maDiv">
  <h1>{{ titre }}<h1>
</div>
```

Listing 3: librairie

```
var app = new Vue({
  el: '#maDiv',
  data: { message: 'Hello World!' }
})
```

2.2 NPM vue-cli

Listing 4: install

```
npm install -g n
n stable
npm i npm@latest -g
npm i -g @vue/cli
npm i -g @vue/cli-init
vue init webpack IMR_2021
cd IMR_2021
npm install
npm run lint
npm run dev
```

3 Documentation

https://v2.vuejs.org/v2/guide/ https://v2.vuejs.org/v2/api/

4 Insertion de données dans l'HTML

JS Basique possible

Listing 5: Vue params in HTML

```
<!-- Réaliser un calcul -->
{{ (2 + 4) * 7 }}

<!-- Appeler des méthodes de string -->
{{ "enssat".toUpperCase() }}

<!-- Opérateurs ternaires -->
{{ 2 > 0 ? 'Toujours' : 'Jamais' }}

<!-- Vue data -->
{{ message }}
```

5 Directives

5.1 v-if, v-else-if, v-else

Permet de créer ou non un élément

Listing 6: v-if

```
<div id="app">
    <section v-if="type␣===␣'default'">
        Le type est default
    </section>

    <section v-else-if="type␣␣===␣'admin'">
        Le type est admin
    </section>
    <section v-else>
        Le type ni default ni admin
    </section>
</div>
```

5.2 v-show

Permet d'afficher ou non un élément (via le style).
Contrairement à un v-if, l'élément est créé dans tous les cas mais n'est affiché que si la condition est vrai.

Listing 7: v-show

```
<div v-show="showModal" class="modal">...</div>
```

5.3 v-for

Listing 8: v-for

```
<ul>
  <li v-for="item␣in␣shoppingCart">
    {{ item.label }} : {{ item.cost }}?
  </li>
</ul>
```

5.4 v-bind

Peut-être pensé comme si la vue est un observeur du modèle.
La vue est "liée" au modèle et suit ses changements.

Listing 9: v-bind

```
<a v-bind:href="item.url">{{ item.name }}</a>
<a :href="item.url">{{ item.name }}</a>
```

5.5 v-on

Gestion des événements

Listing 10: v-on

```
<button v-on:click="alert('Bonjour')">Cliquez ici !</button>
<button @click="alert('Bonjour')">Cliquez ici !</button>
```

5.6 v-model

Permet de lié bilatéralement le modèle et la vue.
Plus souvent utilisé pour le sens vue vers modèle.

Listing 11: v-model

```
<input v-model="texteUtilisateur" placeholder="Modifiez-moi">
<p>Texte utilisateur : {{ texteUtilisateur }}</p>
```

6 Vue

Evitez ici la notation fléchée pour la création de fonctions. Elles ne permettent pas l'access aux données de la Vue.

6.1 Options

data Object | Function

Listing 12: data

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

props Array<string> | Object
Ce sont les paramètres du composant.

Listing 13: Props

```
// simple syntax
Vue.component('props-demo-simple', {
  props: ['size', 'myMessage']
})

// object syntax with validation
Vue.component('props-demo-advanced', {
  props: {
    // type check
    height: Number,
    // type check plus other validations
    age: {
      type: Number,
      default: 0,
      required: true,
      validator: function (value) {
        return value >= 0
      }
    }
  }
})
```

Par exemple, en utilisant v-bind, on peut passer des paramètres à un composant

Listing 14: Props

```
<titre-elem
  v-bind:key="post.id"
  v-bind:title="post.title">
</titre-elem>

<script>
Vue.component("titre-elem", {
        props: ["title"],
        template: "<h3>{{ title }}</h3>"
      });
</script>
```

propsData [key: string]: any
Permet de passer des paramètres à un objet lors de la création (utile pour des tests unitaires, plus rarement en developpement).

Listing 15: propsData

```
var Comp = Vue.extend({
  props: ['msg'],
  template: '<div>{{ msg }}</div>'
})

var vm = new Comp({
  propsData: {
    msg: 'hello'
  }
})
```

computed [key: string]: Function | get: Function, set: Function
Les propriétés compilées ne changent que lorsque une de leurs dépendances est changée.
Elles sont utiles afin de ne pas répéter des calculs quand cela n'est pas nécessaire.

Listing 16: computed

```
var vm = new Vue({
  data: { a: 1 },
  computed: {
    // get only
    aDouble: function () { return this.a * 2},
    // both get and set
    aPlus: {
      get: function () { return this.a + 1 },
      set: function (v) { this.a = v - 1 }
    }
  }
})
```

methods [key: string]: Function
Méthodes liées à l'objet créé. Elle peut accéder à "data" grâce à "this"

Listing 17: methods

```
var vm = new Vue({
  data: { a: 1 },
  methods: {
    plus: function () {
      this.a++
    }
  }
})
vm.plus()
```

watch [key: string]: string | Function | Object | Array
Permet de créé des actions en fonction de changements sur l'élément.

Listing 18: watch

```
var vm = new Vue({
  data: {
    a: 1,
    b: 2,
    c: 3,
    d: 4,
    e: {
      f: {
        g: 5
      }
    }
  },
  watch: {
    a: function (val, oldVal) {
      console.log('new: %s, old: %s', val, oldVal)
    },
    // string method name
    b: 'someMethod',
    // the callback will be called whenever any
    // of the watched object properties change
    // regardless of their nested depth
    c: {
      handler: function (val, oldVal) { /* ... */ },
      deep: true
    },
    // the callback will be called immediately
    // after the start of the observation
    d: {
      handler: 'someMethod',
      immediate: true
    },
    // you can pass array of callbacks,
    // they will be called one-by-one
    e: [
      'handle1',
```

```
      function handle2 (val, oldVal) { /* ... */ },
      {
        handler: function handle3 (val, oldVal) { /* ... */ },
        /* ... */
      }
    ],
    // watch vm.e.f's value: {g: 5}
    'e.f': function (val, oldVal) { /* ... */ }
  }
})
```

el string | Element
Seulement lors de la création d'une nouvelle instance (Ex: new Vue).
contient le chemin vers l'élément source de la vue.

Listing 19: el

```
new Vue({
  el: '#app',
  data: {}
})
new Vue({
  el: document.getElementById('app'),
  data: {}
})
```

template String

Listing 20: template

```
new Vue({
  el: '#app',
  data: {},
  template: "<a></a>"
})
```

render (createElement: () => VNode) => VNode Render a priorité sur template.
Il utilise le DOM virtuel

Listing 21: render

```
new Vue({
  el: '#app',
  data: {t: 'coucou'},
  render: function (createElement) {
    return createElement('h1', this.t)
  }
})
```

mixins Array<Object>
Les "Mixins" permettent de réutiliser des fonctionnalités dans divers composants.
Peut-être assimilé aux "traits"

Listing 22: render

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

// define a component that uses this mixin
var Component = Vue.extend({
  mixins: [myMixin],
  data: function () {
    return {
      message: 'bye',
      bar: 'compo'
    }
})
// data => { message: "bye", foo: "mixin", bar: "compo" }
// In console => 'hello from mixin!'
```

extends Object | Function
Similaire aux mixins, mais correspond plutôt à la notion d'héritage entre classes.

Listing 23: render

```
var CompA = {...};
var CompB = {
  extends: CompA,
  ...
}
```

Listing 24: test bind

```
<html>
<body>
  <div id="app">
    <h2>Date</h2>
    <ul>
      <li>Jour:  {{ jour }}?</li>
      <li>Mois:  {{ mois }}?</li>
      <li>Année: {{ annee }}?</li>
```

```
    </ul>
    <p v-on:enlarge-text="onEnlargeText">
      formatté: {{ formatageDate }}?</p>
    <button v-on:click="$emit('enlarge-text', 0.1)">
      Enlarge text
    </button>
  </div>

  <script>
    const app = new Vue({
      el: '#app',
      data: {
        jour: 6,
        mois: 2,
        annee: 8
      },
      computed: {
        formatageDate() {
          return this.jour + "/" +
                 this.mois + "/" +
                 this.annee
        }
      },
    methods: {
      onEnlargeText: function (e) {
          this.postFontSize += e
        }
      }
    })
  </script>
</body>
</html>
```

7 DOM Virtuel

https://vuejs.org/v2/guide/render-function.html#Nodes-Trees-and-the-Virtual-DOM

8 Composants

Les composants sont des instances de Vue réutilisables.
Préférez les composants monofichiers (comme on avait des classes monofichier en PHP).
L'option "data" d'un Component doit obligatoirement être une fonction contrairement à celle d'une Vue.

Listing 25: Composant 1

```
<div id="components-demo">
  <button-counter title="btn1"></button-counter>
  <button-counter title="btn2"></button-counter>
  <button-counter title="btn3"></button-counter>
</div>

// Define a new component called button-counter
Vue.component('button-counter', {
  props: ['title'],
  data: function () {
    return {
      count: 0
    }
  },
  template: '<button␣v-on:click="count++">You␣clicked␣me␣{{␣count␣}}␣times.</button>'
})
new Vue({ el: '#components-demo' })
```

Listing 26: Component 2

```
<blog-post
  v-for="post␣in␣posts"
  v-bind:key="post.id"
  v-bind:title="post.title"
></blog-post>

Vue.component('blog-post', {
  props: ['title'],
  template: '<h3>{{␣title␣}}</h3>'
})

new Vue({
  el: '#blog-post-demo',
  data: {
    posts: [
      { id: 1, title: 'My␣journey␣with␣Vue' },
      { id: 2, title: 'Blogging␣with␣Vue' },
      { id: 3, title: 'Why␣Vue␣is␣so␣fun' }
    ]
  }
})

new Vue({ el: '#components-demo' })
```

8.1 slot

Permet de passer du contenu dans un "template", l'élément est remplacé par le contenu.

Listing 27: Slot

```
<alert-box>
  Something bad happened.
</alert-box>

Vue.component('alert-box', {
  template: `
    <div class="demo-alert-box">
      <strong>Error!</strong>
      <slot></slot>
    </div>
  `
})
```

8.2 Single-file components

Listing 28: Single File

```
<template>
  <div class="checkbox-wrapper" @click="check">
    <div :class="{ checkbox: true, checked: checked }"></div>
    <div class="title">{{ title }}</div>
  </div>
</template>
<script>
export default {
  data() {
    return { checked: false, title: 'Check me' }
  },
  methods: {
    check() { this.checked = !this.checked; }
  }
};
</script>
```

9 Exemple

https://codesandbox.io/s/github/vuejs/vuejs.org/tree/master/src/v2/examples/vue-20-dynamic-components