VUE JS

VUE.JS NOTIONS AVANCÉES

RAPPELS : VUE

Reactive Components for Modern Web Interfaces
V de MVC : Micro-framework de présentation
Full JavaScript (donc compatible : jusque IE9)
Extensible (avec les plugins de vue ou d'autres)

RAPPELS : COMPOSANT

```
Vue.component('fct-component', {
    template: '',
    data() {
        return {
            // propriétés internes
            // elles seules peuvent muter dans ce composant
        }
    },
    props: {
        // contrat interface d'entrée
    },
    methods: {
        // Fonctions utilisée en interne
        someFunction() {
            //contrat interface sortie
            this.$emit('topic', optionalParams)
        }
    },
    computed: {
        // Props transformées
        // ES6 get/set pour composant vues
    },
    //...
})
```

RAPPELS : COMPOSANT

```
Vue.component('fct-componont', {
    //...
    components: {
        // déclaration de composant locaux à ce composant
    },
    filters: {
        // déclaration de 'filtres' locaux
    },
    directives: {
        // déclaration de directives locales
    },
    created() { },
    mounted() { },
    updated() { },
    destroyed() { },
    mixins: []
})
```

RAPPELS : TEMPLATE

```
<template>
        <span>{{ number + 1 }}</span>
        <div :id="'item-' + id"></div>
        <div v-html="inner_html"></div>
        <span v-once>{{ amount | capitalize | currency '€' }}</span>
        <p v-if="greeting">Hello!</p>
        <input type ="text" v-model="searchFilter"/>
        <ul>
                <li v-for="item in items | filterBy searchFilter">
                        {{ item.attr }}
                </li>
        </ul>
        <a v-bind:href="url"></a>
        <a :href="url"></a>
        <a v-on:click="doSomething">
        <a @click="doSomething">
        <a @click.stop.prevent="doSomething">
        <input @keyup.enter="onEnter"><input @keyup.13="onEnter">
        <div :class="[classA, classB]"></div>
</template>
```

RAPPELS : DIRECTIVE

Vue.js fournit un ensemble de directives (pseudo-attribut html) pour structurer l'application

v-text : met à jour innerText
v-html : met à jour innerHTML (parse la chaine attention à l'XSS)
v-show : agit sur l'attribut display de l'élément
v-if : ajoute le template conditionnellement
v-else : ajoute le template conditionnellement (requiert v-if)
v-else-if : ajoute le template conditionnellement (requiert v-if)
v-for : ajoute le template itérativement sur une collection
v-on : vue en détail dans le chapitre templating
v-bind : vue en détail dans le chapitre templating
v-model : vue en détail dans le chapitre templating
v-pre : annule la compilation du template (pas de parse)
v-cloak : directive présente avant la phase compile
v-once : invoque le render 1 seule fois (n'écoute pas les changements de donnée)

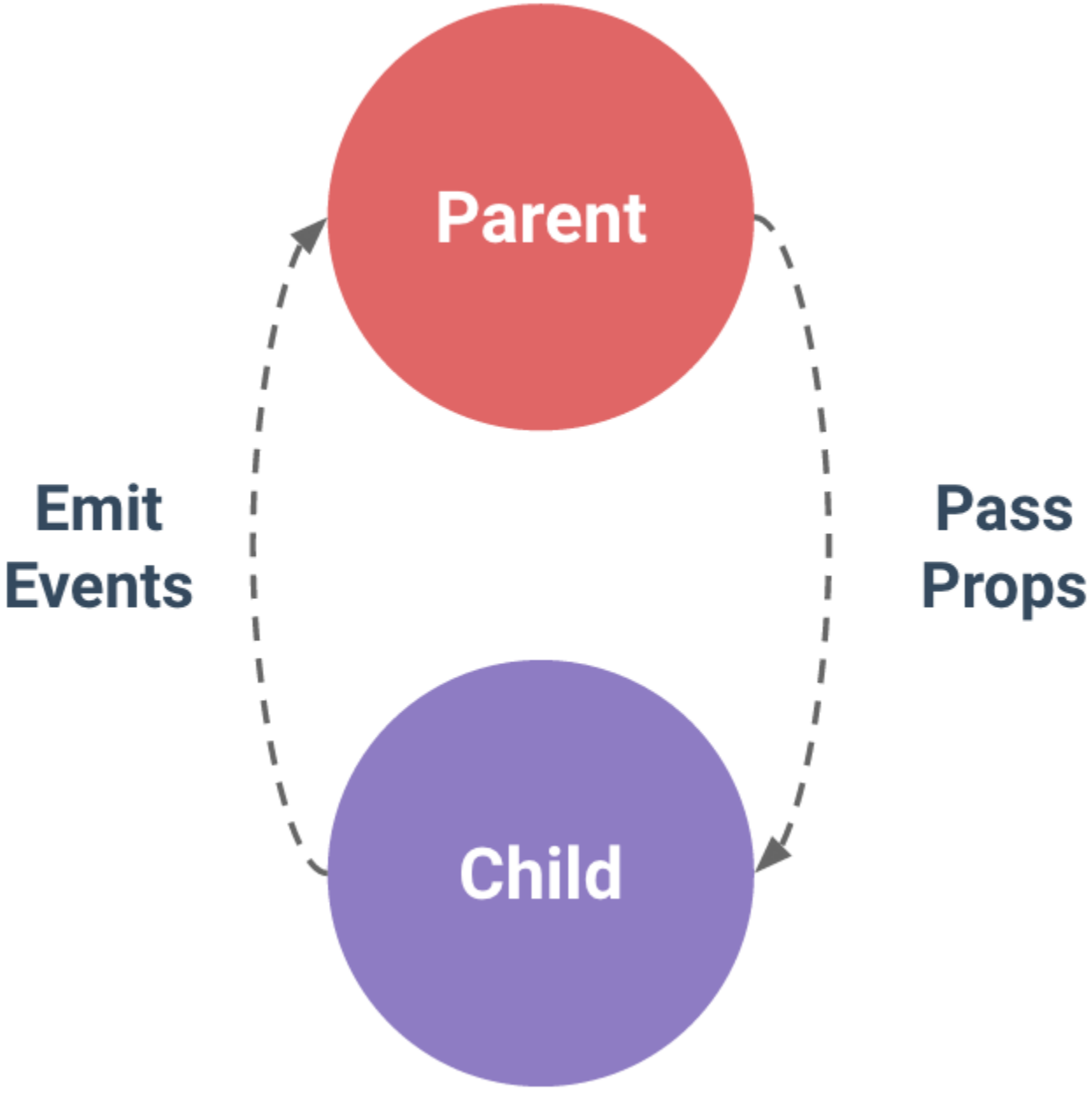
RAPPELS : PROPS

```
Vue.component('fct-component', {
    props: {
        // basic type check (`null` means accept any type)
        propA: Number,
        // multiple possible types
        propB: [String, Number],
        // a required string
        propC: {
            type: String,
            required: true
        },
        // a number with default value
        propD: {
            type: Number,
            default: 100
        },
        // object/array defaults should be returned from a
        // factory function
        propE: {
            type: Object,
            default: function () {
                return { message: 'hello' }
            }
        },
        // custom validator function
        propF: {
            validator: function (value) {
                return value > 10
            }
        }
```

```
        }

})
```

RAPPEL : COMMUNICATION


Parent
Emit
Events
Pass
Props
Child

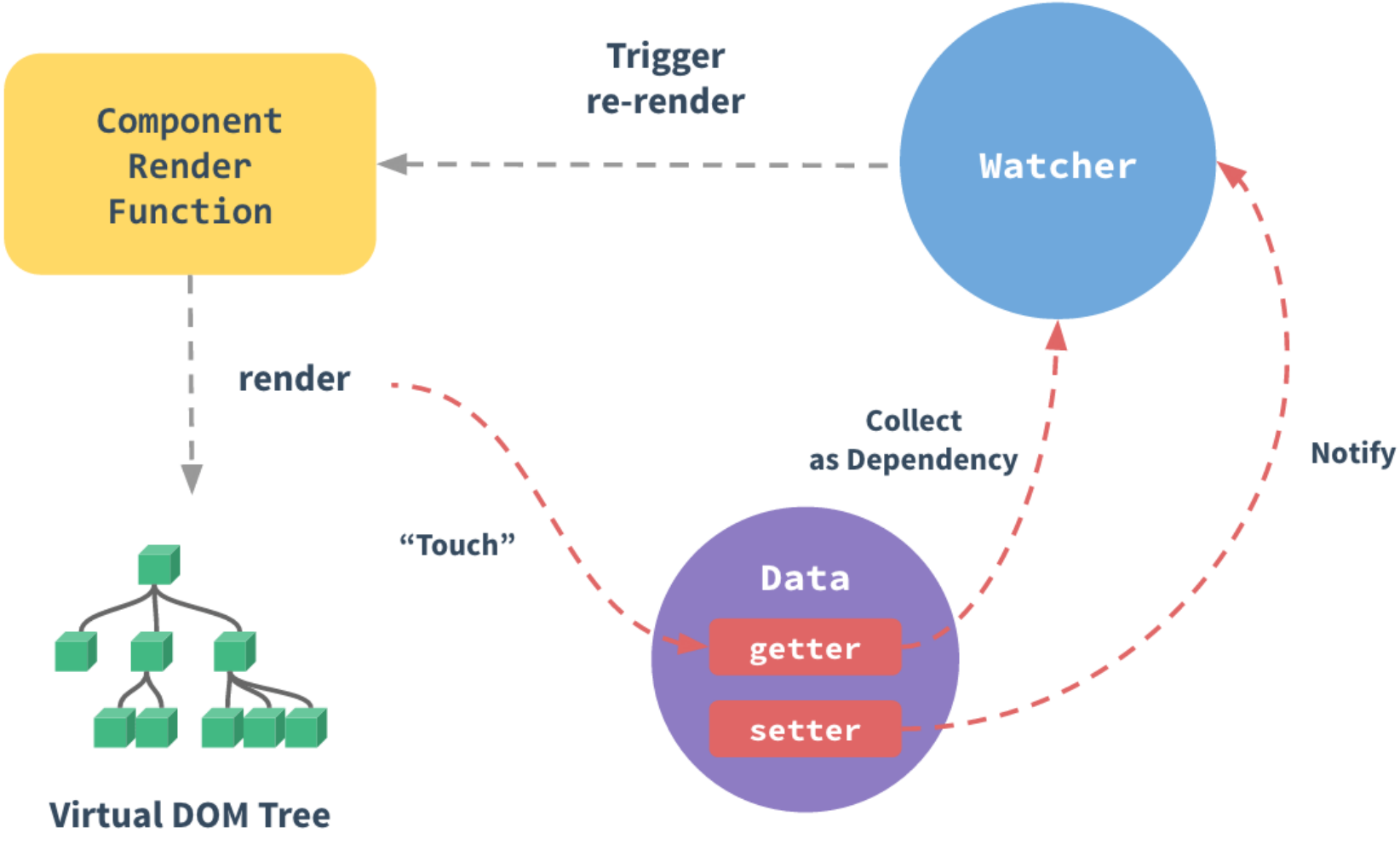
NOTIONS AVANCÉES

API

En plus de component et filter, Vue.js expose un ensemble de méthodes utilitaires pour :

- Gérer la détection du changement `set`, `delete`
- créer des directives : `directive`
- déclarer l'utilisation d'un plugin : `use`
- compiler un template html : `compile`
- 'hériter' des comportements d'une instance à une autre : `mixin`
- lancer une fonction asynchrone : `nextTick`

DÉTECTION DU CHANGEMENT



LIMITATIONS

- Vue observe UNIQUEMENT get & set
- La conversion est faite uniquement à l'instanciation

VIRTUAL-DOM

- Vue utilise le virtual DOM pour mettre à jour le DOM réel de façon asynchrone
- Cela limite le nombre de manipulations DOM & de calculs (performance)
- Cela complique la compréhension en debug (data mise à jour, mais rendue ultérieurement)
- Si vous avez besoin d'interagir avec le DOM vous avez besoin de `nextTick`

```
var vm = new Vue({
    el: '#example',
    data: {
        message: '123'
    }
})
vm.message = 'new message' // change data
vm.$el.textContent === 'new message' // false
Vue.nextTick(function () {
    vm.$el.textContent === 'new message' // true
})
```

LE CAS DES OBJETS

- Ce cas n'est pas détecté :

```
myComp.propObject.subProp = newValue
```

C'est là que Set/Delete rendent service :

```
Vue.set(myComp.propObject, 'subProp', 2)

//Alias $set
this.$set(this.propObject, 'subProp', 2)
```

LE CAS DES TABLEAUX

- Ce cas n'est pas détecté :

```
myComp.items[indexOfItem] = newValue
```

Vue fournit en plus de Set des méthodes de Mutations pour ses propriétés tableaux :

- push()
- pop()
- shift()
- unshift()
- splice()
- sort()
- reverse()

EXEMPLE

```
myComp.items[indexOfItem] = newValue // pas de détection

// Vue.set
Vue.set(myComp.items, indexOfItem, newValue) // comme pour les objets

// Array.prototype.splice
myComp.items.splice(indexOfItem, 1, newValue) // Mutation method

// Array.prototype.push
myComp.items.push(newValue) // Mutation method pour l'ajout
```

CRÉATION DE DIRECTIVE

```
Vue.directive('focus', {
    // When the bound element is inserted into the DOM...
    inserted: function (el) {
        // Focus the element
        el.focus()
    }
})
```

POSSIBILITÉS DE DIRECTIVES

- bind: Appelé 1 seule fois au moment du lien entre la directive et son composant
- inserted: Appelé à l'insertion dans un noeud parent (pas nécessairement dans le DOM)
- update: appelé à chaque changement du noeud lié (appelé même si a directive n'a pas changé)
- componentUpdated: appelé après que le composant lié et tous ses enfants aient été mis à jour
- unbind: Appelé 1 seule fois au retrait de la directive

HOOK SUR LE CYCLE DE VIE

- beforeCreate : avant la lecture de data, et le branchement des watchers de props
- created : avant l'ajout au v-dom, apres la lectude de data, props, methods, computed
- beforeMount : le tempalte est compilé mais non injecté
- mounted : le template est injecté
- beforeUpdate : phase de détection du changement, avant le changement
- updated : phase de détection du changement, après le changement
- beforeDestroy : juste avant la destruction, instance 100% opérationnelle
- destroyed : composant `unmounted`, observer détruits, composants fils détruits

MIXINS

- Vue.js fournit la réutilisation de code par **`mixin`**
- un Mixin Vue.js respecte l'interface d'une instance Vue
- un Mixin est un simple **`Object Literal`**
- un mixin est attaché à une instance Vue par la propriété mixins
- Le mixin est mergé **`intelligemment`**
  - les hooks sont enregistrés en tableau (tous éxécutés)
  - les props/methods sont mergés en 1 seul objet (priorité au composant)

EXEMPLE

```
// define a mixin object
var myMixin = {
    created () {
        console.log('created from mixin')
    },
    methods: {
        isValid () {
            console.log('Am i valid ?')
        },
        done () {
            console.log('mixin done')
        }
    }
}
// define a component that uses this mixin
Vue.component('extended-comp', {
    mixins: [myMixin],
    methods: {
        done() {
            console.log('component done')
        }
    },
    created () {
        console.log('created from component')
    }
})
```

POUR ALLER PLUS LOIN

- Gestion des transitions
- Fonction de programmation des template
- Création/Manipulation de plugins

EN RÉSUMÉ

- Sous ses apparences simple, Vue.js offre des possibilités avancées
- La détection du changement handicape le debug
- Vue permet l'héritage de comportement/propriétés via les Mixins

ROUTAGE

PHILOSOPHIE

Le routage est le mécanisme par lequel des chemins sont sélectionnés dans un réseau pour acheminer les données d'un expéditeur jusqu'à un ou plusieurs destinataires.

Dans un `navigateur` la `navigation` est naturelle, chaque URL est une route, et les liens de nos applications constituent le routage.

VUE-ROUTER

`vue-router` est un plugin officiel de `vue.js`, c'est à dire qu'il est maintenu par la même équipe que la librairie.

Il s'intègre parfaitement aux applications et composants vue.js, et embarque des fonctionnalités similaires aux concurrents.

BASE

```
import Vue from 'vue'
import VueRouter from 'vue-router'
Vue.use(VueRouter)

import App from './App'

const routes = [
    { path: '/foo', component: Foo },
    { path: '/bar', component: Bar }
]

const router = new VueRouter({
    routes //routes est le tableau des routes (configuration du router)
})

const app = new Vue({
    el: '#app',
    router,
    render: h => h(App)
})
```

Prêt à router la navigation de notre Application !

AFFICHER LE COMPOSANT

Ce n'est pas le tout d'avoir des routes, encore faut-il dire à l'application où insérer le composant.

Pour ce faire VueRouter fournit le composant **`RouterView`** importée par le **`Vue.use(VueRouter)`**, il suffit donc d'insérer dans le template de App.vue :

```
<!-- Routed views go here -->
<router-view></router-view>
```

NAVIGUER

Pour naviguer de vue en vue il faut activer une Route :

- En saisissant l'URL à la main
- En activant une directive **`RouterLink`**
- Une classe CSS peut-être associée à **`RouterLink`** via la directive **`active-class`**

```
<h1>Vue Router</h1>
    <nav>
        <router-link to="" active-class="selection">Accueil</router-link>
        <router-link to="/users" active-class="selection">Utilisateurs</router-l
    </nav>
    <router-view></router-view>
```

PASSER DES PARAMÈTRES

Il suffit d'utiliser `:` dans la description de la route pour chaque paramètre nommé :

```
const User = {
    template: '<div>Utilisateur</div>'
}

const router = new VueRouter({
    routes: [
        // Les segments dynamiques commencent avec la ponctuation deux-points
        { path: '/utilisateur/:id', component: User }
    ]
})
```

On peut utiliser les paramètres dans le composant **`user`** grâce à la propriété **`this.$route.params`**

```
const User = {
    template: '<div>Utilisateur : {{$route.params.id}}</div>'
}
```

PRIORITÉS

Une même URL peut activer plusieurs routes, dans ce cas la priorité est affecté à la première route déclarée dans le tableau de configuration.

```
const routes = [
    { path: '/user', component: Users },
    { path: '/user/:id', component: UserDetail }
]

const routes = [
    { path: '/user/:id', component: UserDetail },
    { path: '/user', component: Users }
]
```

ROUTE-CEPTION

Le fonctionnement du router de vue permet d'avoir des routes imbriquées.

```
const routes = [
    {
        path: '/user/:id',
        component: User,
        children: [
            {path: 'edit', Component: UserForm},
            {path: '', Component: UserDetail}
        ]
    }
]
```

Ici le composant **`User`** chargé dans le **`router-view`** d'App.vue contient lui-même un composant **`router-view`** pour recevoir **`UserDetail`** ou **`UserForm`** en fonction de la suite de la route '/user/:id' ('' ou bien '/edit').

AUTRES PARAMÈTRES DU ROUTER

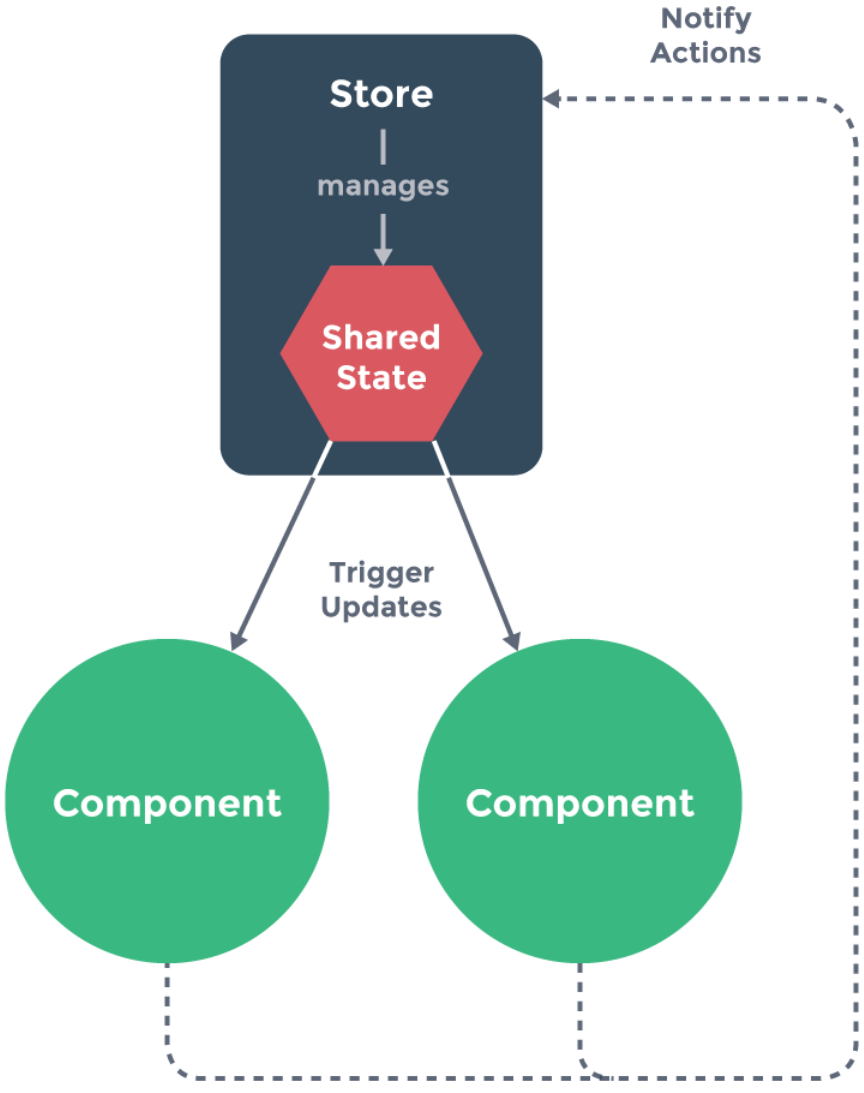
- mode : "hash" | "history" | "abstract"
- base : '/', URL de départ de l'application pour les cas de déploiements dans domain.name/app/
- linkActiveClass : nom de remplacement de router-link-active

AUTRES OPTIONS DES ROUTES

- name: nom de la route pour pouvoir naviguer en utilisant le nom plutôt que le path
- components: { [name: string]: Component }; Pour charger de multiples composants pour une même route
- redirect: redirection d'une route vers une autre
- alias: autres nom pour un même résultat (2 routes chargeant la même vue, mais sans appliquer de redirection)
- props: pour transmettre les paramètres de la route aux props du composant (pas besoin d'utiliser $route.params)

GESTION D'ÉTAT

STATE MANAGER PATTERN


Notify
Actions
Store
manages
Shared
State
Trigger
Updates
Component
Component

QUAND L'UTILISER ?

- Ce pattern complexifie l'architecture de base
- Ce pattern simplifie le développement et le debug des grosses applications

REDUX

- Implémentation de Flux pour React
- Utilisable avec d'importe quel FWK d'application composant
- Alternative sérieuse à Vuex

VUEX

- Implémentation de Flux pour Vue uniquement
- Vuex connais l'existence de Vue et y est couplé
- En contrepartie l'interaction avec les composants est intuitive et simple

STORE

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

const store = new Vuex.Store({
    state: {
        count: 0
    },
    getters: {
        double: state => {
            return state.count * 2
        }
    },
    mutations: {
        increment (state) {
            state.count++
        }
    },
    actions: {
        incrementAsync (context) {
            setTimeout(() => context.commit('increment'), 0)
        }
    }
})

export default store
```

MAIN.JS

```
import Vue from 'vue'

import store from './store'
import App from './App'

new Vue({
    el: '#app',
    store,
    render: h => h(App)
})
```

COMPOSANT

```
<div>{{ $store.state.count }}</div>
<div>{{ double }}</div>
<button @click="increment"></button>

<script type="text/javascript">
export default {
    computed: {
        double () {
            return this.$store.getter.double
        }
    },
    methods : {
        increment() {
            this.$store.dispatch('incrementAsync')
        }
    }
}
</script>
```

MUTATIONS PARAMÉTRÉES

```
// ...
mutations: {
    increment (state, n) {
        state.count += n
    }
}
//...
store.commit('increment', 10)
//...
actions: {
    incrementAsync (context) {
        setTimeout(() => context.commit('increment'), 0)
    }
}
//...
store.dispatch('incrementAsync', 10)
```

EN RÉSUMÉ

- Si votre application est grosse, utiliser un State Manager
- Redux est l'implémentation de référence
- Vuex est spécifiquement lié à Vue
- Ne pas oublier d'affecter le store à l'application

BEST PRACTICES

PERFS : V-PRE & V-ONCE

- Pour les templates statiques, ou ceux sortant de la détection du changement
- V-pre = no parsing
- V-once = évaluer(parser) 1 seule fois (pas de détection du changement)

CODER

- Préférer des déclarations locales
  - build plus petit
  - Lazy Loadinf
- KISS : les composants doivent être le plus élémentaires possible
- Never Touch the DOM
- Eviter l'ajout de watch

EN RÉSUMÉ

- Approche composant complète
- Apprentissage Facile
- Super léger
- Totalement Flexible

RESSOURCES

- https://vuejs.org
- https://www.getrevue.co/profile/vuenewsletter
- https://madewithvuejs.com/
- https://www.mindmeister.com/842448234/vue-js