Formation

Vue.js

m2iformation.fr

Sommaire

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Introduction

Qu'est-ce que Vue ?

- Vue (/vjuː/ à prononcer comme en anglais: view) est un framework JavaScript qui se repose sur les standards HTML, CSS et JavaScript
- Il propose une manière efficace de déclarer des composants pour la construction d'interfaces utilisateur, qu'elles soient simples ou complexes

Utilisation de Vue

- Extension du HTML statique sans étape de construction
- Intégration de Web Components (éléments personnalisés) sur n'importe quelle page
- Application mono-page (SPA)
- Fullstack / Rendu côté serveur (SSR)
- JAMStack / Génération de sites statiques (SSG)
- Adapté pour l'ordinateur de bureau, le mobile, pour le WebGL et même le terminal

Single-File Components (SFC)

Un SFC Vue, comme son nom l'indique, encapsule la logique (**JavaScript**), le modèle (**HTML**) et les styles (**CSS**) du composant dans un seul fichier

```
<script setup>
</script>

<template>
</template>

<style scoped>
</style>
```

Styles d'API

Les composants Vue peuvent être créés dans deux styles d'API différents :

1. **Options API** : permet de définir la logique d'un composant en utilisant un objet d'option (`data` , `methods` , `mounted`). Les propriétés sont exposées sur `this` dans les fonctions
2. **Composition API**: permet de définir la logique d'un composant à l'aide des fonctions API importées

l'Options API est implémentée par dessus la Composition API, les concepts sont partagés entre les deux styles.

Création d'une application Vue.js

Exécuter la commande suivante dans votre invite de commandes :

```
npm create vue@latest
```

```
cd {nom du projet}
```

```
npm install
```

```
npm run dev
```

Éditeur et extension

La documentation de Vue.v recommande l'utilisation de VSCode ainsi que l'extension Volar (Vue Language Features)

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Créer une application

Instance d'application

- Toute application Vue commence par créer une instance d'application avec la function createApp

```
import { createApp } from "vue";
const app = createApp({
  // Composants racines
});
```

Composant racine (Root component)

- L'objet passé à createApp est un composant qui contient d'autres composants enfants

```
import { createApp } from "vue";
import App from "./App.vue";

const app = createApp(App);
```

Monter l'application

- Une instance d'application ne rendra rien tant que sa méthode `.mount()` n'aura pas été appelée
- Elle attend un argument "container", qui peut être soit un élément DOM réel, soit une chaîne de sélection

```
<div id="app"></div>
```

```
app.mount("#app");
```

Configuration de l'application

- L'instance d'application expose un objet `.config` qui nous permet de configurer quelques options au niveau de l'Application
- Exemple en définissant un gestionnaire d'erreurs au niveau de l'application qui capture les erreurs de tous les composants descendants

```
app.config.errorHandler = (err) => {
  /* handle error */
};
```

Multiples instances d'applications

La fonction `createApp` permet de créer plusieurs instances d'applications Vue sur la même page avec son propre scope et sa configuration

```
const app1 = createApp({
  /* ... */
});
app1.mount("#container-1");

const app2 = createApp({
  /* ... */
});
app2.mount("#container-2");
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Syntaxe de template

Template

- Vue utilise une syntaxe de template basée sur HTML
- Vue compile les templates en code JavaScript hautement optimisé
- Combiné avec le système de réactivité, Vue est capable de déterminer intelligemment le nombre minimal de composants à restituer et d'appliquer la quantité minimale de manipulations DOM lorsque l'état de l'application change

Interpolation de texte

- L'interpolation de texte se fait à l'aide de double accolades `{{}}`
- La propriété **msg** sera mise à jour chaque fois que sa valeur change

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

HTML Brut

- Pour rendre du text HTML brut il faut utiliser la **directive** `v-html`

```
<p>Using v-html directive: <span v-html="rawHtml"></span></p>
```

- Les directives sont préfixées par `v-` pour indiquer qu'il s'agit d'attributs spéciaux fournis par Vue
- ⚠️ L'affichage dynamique de code HTML peut être dangereux en raison de failles XSS

Liaisons d'attributs

- Les accolades ne peuvent pas être utilisées dans les attributs HTML
- La directive `v-bind` permet d'utiliser des propriétés dynamiques sur des attributs

```
<div v-bind:id="dynamicId"></div>
```

- Si la valeur vaut `null` ou `undefined` alors l'attribut est supprimé
- Raccourci: `:`

```
<div :id="dynamicId"></div>
```

Attributs booléens

- v-bind peut être utilisé dans le cas ou un attribut se comporte comme un booléen

```
<button :disabled="isButtonDisabled">Button</button>
```

Liaison dynamiques de plusieurs attributs

- Les objets peuvent être utilisés pour liers plusieurs attributs à un élément HTML

```
const objectOfAttrs = {
  id: "container",
  class: "wrapper",
};
```

```
<div v-bind="objectOfAttrs"></div>
```

- Une fois rendu l'élément HTML ressemblera à ça :

```
<div id="container" class="wrapper"></div>
```

Expressions javascript

Les templates vue permettent d'utiliser les expressions JavaScript dans les cas suivants :

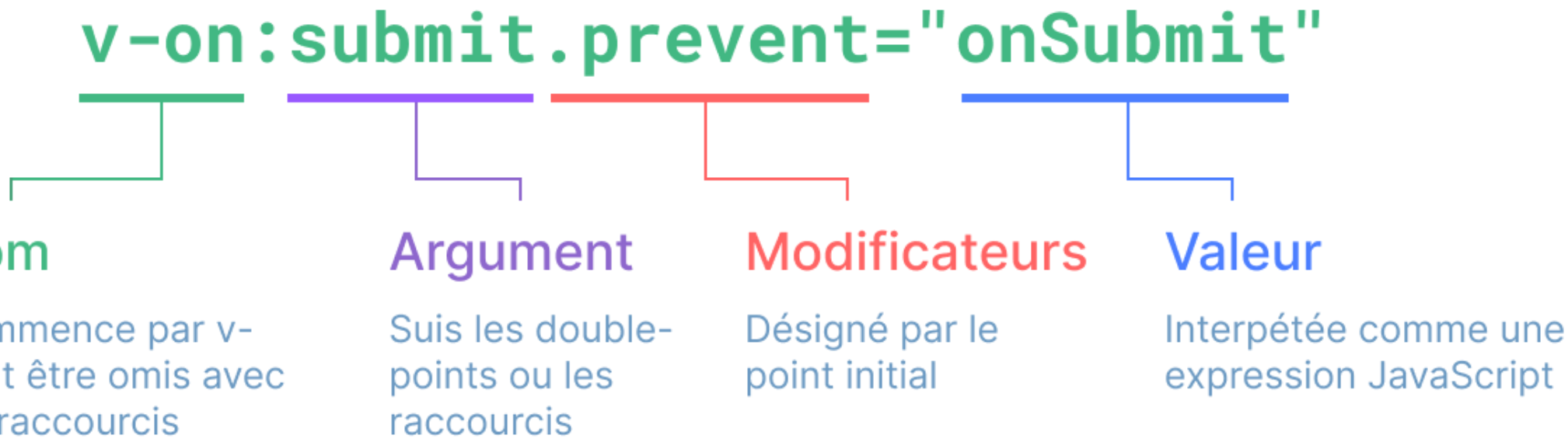
- Dans une interpolation de texte `{{}}`
- Dans les directives Vue `v-`

```
{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}

<div :id="`list-${id}`"></div>
```

Directives

- Les directives sont des attributs spéciaux préfixés par `v-`
- Certaines directives peuvent avoir des arguments:
  - `vbind:href="url"`
  - `v-on:click="doSomething"`
- Il est possible d'utiliser des arguments dynamiques avec `[]`
  - `<a v-bind:[attributeName]="url"> ... </a>`
- Les modificateurs sont des suffixent qui rajoutent un comportement spécial à la directive : `@submit.prevent="onSubmit"`

Syntaxe de directive



UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Fondamentaux de la réactivité

Qu'est-ce que la réactivité

- La réactivité permet de **détecter automatiquement un changement de valeur** d'une variable pour mettre à jour le DOM.
- Ce mécanisme est rendu possible grâce au système de suivi des dépendances.
- Globalement, Vue analyse les entrées/sorties réalisées sur la variable pour déclencher des évènements.

État réactif

- Pour déclarer un état réactif il est nécessaire d'utiliser la fonction `ref()`
- `ref()` prend l'argument et l'enveloppe dans un objet ref avec une propriété `.value`
- Une ref peut prendre **n'importe quel type de valeur** et rendre sa valeur profondément réactive (objet, tableaux, Map, ...)

```
import { ref } from "vue";
const count = ref(0);
console.log(count); // { value: 0 }
console.log(count.value); // 0
```

Utilisation de la fonction setup()

L'utilisation d'un état réactif dans le template nécessite l'utilisation de la fonction `setup()`

```
export default {
  // Hook de la compositionAPI
  setup() {
    const count = ref(0);

    // expose l'état au template
    return {
      count,
    };
  },
};
```

```
<div>{{ count }}</div>
```

Pas besoin d'ajouter `.value`, l'objet ref est automatiquement déballé dans le template

<script setup>

- L'utilisation de composant monofichiers (SFC) permet d'utiliser un raccourci pour export l'état les méthodes

```
<script setup>
import { ref } from 'vue'
...
const count = ref(0)
const increment = () => count.value++;
</script>

<template>
  <button @click="increment"> {{ count }} </button>
</template>
```

reactive()

- `reactive()` permet de rendre un objet réactif
- Les objets réactifs sont des proxys JavaScript
- Les objets imbriqués par réactive deviennent eux même réactif
- `ref()` utilise également reactive lorsque la valeur passée est un objet
- ⚠️ l'utilisation de la destructuration fait perdre la réactivité de l'objet

```
import { reactive } from "vue";
const state = reactive({ count: 0 });
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Propriétés calculées

computed()

- Lorsque certaines calculent impliquent des données réactive, il est intéressant d'utiliser des propriétés calculées
- Ces références calculées nécessitent l'utilisant de la fonction `computed()`
- Le résultat de cette fonction renvoie un objet avec une propriété `.value()`

```
const publishedBooksMessage = computed(() => {
  return author.books.length > 0 ? "Yes" : "No";
});
```

Avantage des propriétés calculées

- Les propriétés calculées ne sont réévaluées que lorsqu'une dépendance réactive change
- La valeur de la propriété calculée est conservée dans le cache ce qui évite de ré-exécuter des traitements inutiles
- ⚠️ Les propriétés ne doivent pas faire de requêtes asynchrones ou de manipulation du DOM

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Liaison de classes et de styles

Liason de style par objet

Vue fourni des améliorations lorsque `v-bind` est utilisée avec les `class` ou le `style`

```
<div :class="{ active: isActive, 'text-danger': hasError}"></div>
```

- Utilisation des propriétés calculées:

```
const classObject = computed(() => ({
  active: isActive.value && !error.value,
  "text-danger": error.value && error.value.type === "fatal",
}));
```

```
<div :class="classObject"></div>
```

Liaison par tableau

La liaison de classe peut également se faire par tableaux

```
const activeClass = ref("active");
const errorClass = ref("text-danger");
```

```
<div :class="[activeClass, errorClass]"></div>
```

Les deux syntaxes peuvent être combinées :

```
<div :class="[{ active: isActive }, errorClass]"></div>
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Rendu conditionnel

v-if v-else

- La directive `v-if` est utilisée pour restituer conditionnellement un bloc
- Le bloc ne sera rendu que si l'expression de la directive retourne une valeur évaluée à vrai
- `v-else` pour indiquer le bloc "sinon"

```
<button @click="awesome = !awesome">Basculer</button>

<h1 v-if="awesome">Vue is awesome!</h1>
<h1 v-else>Oh no 😢</h1>
```

v-if v-else-if

```
<div v-if="type === 'A'">
  A
</div>
<div v-else-if="type === 'B'">
  B
</div>
<div v-else>
  C
</div>
```

v-if avec <template>

- `v-if` ne fonctionne qu'avec un élément HTML
- Pour rendre un bloc invisible il faut utiliser cette directive sur un élément `<template>`

```
<template v-if="ok">
  <h1>Title</h1>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</template>
```

v-show

- `v-show` permet de basculer la propriété css display de l'élément
- v-show est à préférer à v-if quand un élément à besoin de basculer régulièrement car est plus optimisé

```
<h1 v-show="ok">Hello!</h1>
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Rendu de liste

v-for

- La directive `v-for` permet de rendre une liste d'éléments basé sur un tableau

```
<li v-for="item in items">
  {{ item.message }}
</li>
```

- Utilisation de l'index dans la directive

```
<li v-for="(item, index) in items">
  {{ parentMessage }} - {{ index }} - {{ item.message }}
</li>
```

Utilisation de clé

- Pour assurer un ordre dans l'affichage des éléments, il est important de fournir un attribut `key` uniquement pour chaque item
- Cette clé est recommandé lorsque le contenu du DOM itéré comporte des composants ou un élément avec un état

```
<div v-for="item in items" :key="item.id">
  <!-- contenu -->
</div>
```

Détection de changements

- Les méthodes qui modifient les tableaux sont réactives: `push`, `pop`, `sort` ...
- Les méthodes qui créent une nouvelles instances de tableaux doivent être traitées comme suit :

```
items.value = items.value.filter((item) => item.message.match(/Foo/));
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Gestion des évènements

Écouter un évènement

- Il est possible d'écouter des évènements avec la directive `v-on` et son raccourci `@`
- Exemple d'évènement: `v-on:click="handler"` ou `@click="handler"`
- Il est possible d'utiliser le gestionnaire inline:

```
<button @click="count++">Add 1</button>
```

- Ou d'utiliser la référence de la fonction:

```
<button @click="greet">Greet</button>
```

Modificateurs d'évènements

Il est possible d'avoir besoin d'utiliser certaines méthodes liés à l'évènement. Pour cela on peut utiliser les différentes méthodes suivantes :

- `.stop` : arrêter la propagation de l'event
- `.prevent` : annule le comportement par défaut
- `.self` : déclenche l'event que sur lui même
- `.capture`
- `.once`
- `.passive`

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Les formulaires

Liaison de données

Lors de la création de formulaire, il est souvent nécessaire de devoir lier la valeur saisie avec une variable réactive.
La première méthode pour y arriver consisterai à faire:

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

Une version simplifiées avec `v-model` nous permet d'avoir le même résultat:

```
<input v-model="searchText" />
```

CheckBox

- La variable check est un booléen:

```
<input type="checkbox" id="checkbox" v-model="checked" />
<label for="checkbox">{{ checked }}</label>
```

- Un tableau permet également de récupérer plusieurs valeurs:

```
const checkedNames = ref([]);
<input type="checkbox" id="jack" value="Jack" v-model="checkedNames">
<label for="jack">Jack</label>

<input type="checkbox" id="john" value="John" v-model="checkedNames">
<label for="john">John</label>
```

Radio button

- La valeur de l'élément HTML est récupérée via le v-model

```
<div>Picked: {{ picked }}</div>

<input type="radio" id="one" value="One" v-model="picked" />
<label for="one">One</label>

<input type="radio" id="two" value="Two" v-model="picked" />
<label for="two">Two</label>
```

Select

- Exemple de template de sélection:

```
<div>Selected: {{ selected }}</div>

<select v-model="selected">
  <option disabled value="">Please select one</option>
  <option>A</option>
  <option>B</option>
  <option>C</option>
</select>
```

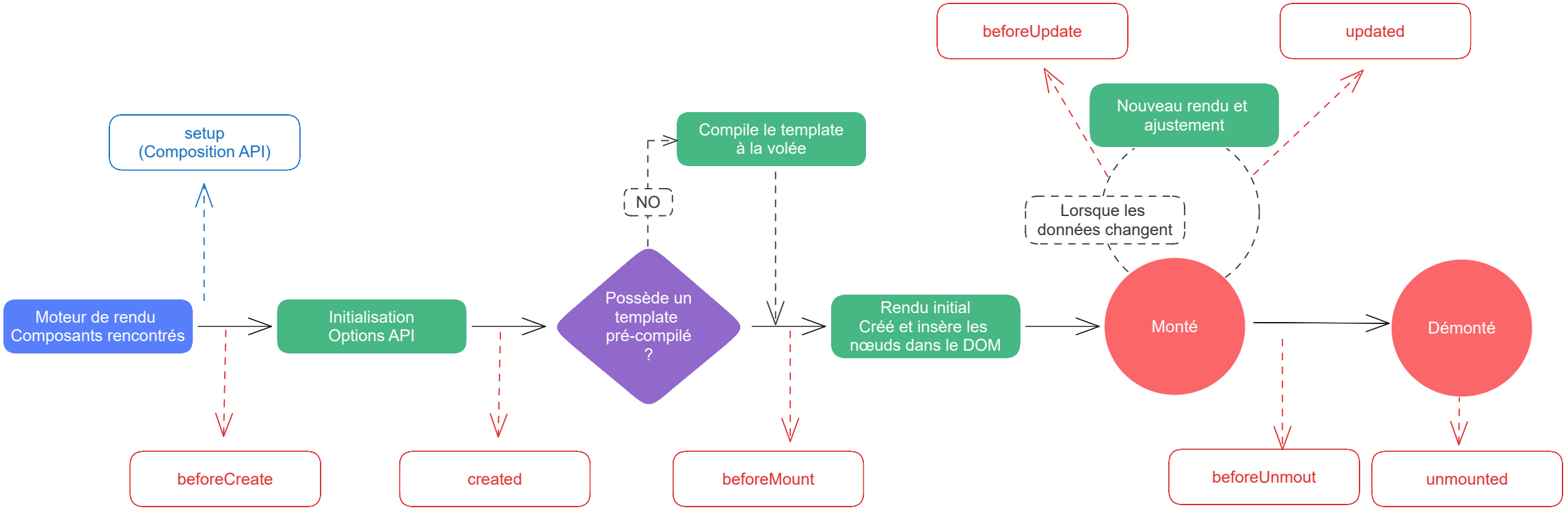
UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Les hooks du cycle de vie

Fonctionnement

- Chaque instance d'un composant Vue passe par **une série d'étapes d'initialisation** lorsqu'elle est créée : observation des données, compilation du templates, monter l'instance sur le DOM ...
- En cours de route, des fonctions appelées hooks du cycle de vie sont également exécutées, donnant la possibilité à l'utilisateur **d'ajouter son propre code à des étapes spécifiques**

Diagramme des hooks



Utilisation d'un hook

Pour utiliser l'un des hooks d'un composant, il suffit de l'importe depuis Vue :

```
<script setup>
  import {onMounted} from 'vue' onMounted(() =>{" "}
  {console.log(`the component is now mounted.`)})
</script>
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Observateurs

Définition

- Un observateur une fonction qui est appelée chaque fois qu'une propriété de l'objet observé change
- Les observateurs sont utiles pour effectuer des actions en réponse à des changements de propriétés, comme la mise à jour de l'interface utilisateur
- Les observateurs peuvent être définis sur des propriétés de données, des propriétés calculées, des propriétés de composant, etc

watch()

- La fonction `watch` permet de déclencher une callback chaque fois qu'une partie d'un état réactif change
- Son premier argument est une source réactive: une ref, un object réactif, une fonction accesseur ou un tableau

```
watch(x, (newX) => {
  console.log(`x is ${newX}`);
});
watch(
  () => x.value + y.value,
  (sum) => console.log(sum)
);
```

Spécificité de watch

Il n'est pas possible d'observer la propriété d'une objet réactif par son accesseur classic

❌ cette forme ne fonctionne pas

```
watch(obj.count, (count) => {
  console.log(`count is: ${count}`);
});
```

✅ Il faut utiliser une fonction accesseur

```
watch(
  () => obj.count,
  (count) => {
    console.log(`count is: ${count}`);
  }
);
```

watchEffect

- `watchEffect` exécute immédiatement une fonction tout en suivant de manière réactive ses dépendances et la ré-exécute dès que les dépendances sont modifiées
- watchEffect n'a pas besoin de spécifier explicitement sa source

```
watchEffect(async () => {
  const response = await fetch(
    `https://jsonplaceholder.typicode.com/todos/${todoId.value}`
  );
  data.value = await response.json();
});
```

watch vs watchEffect

watch

- traque seulement la source explicitement observée
- Le rappel n'est déclenché que lorsque la source a bien changé

watchEffect

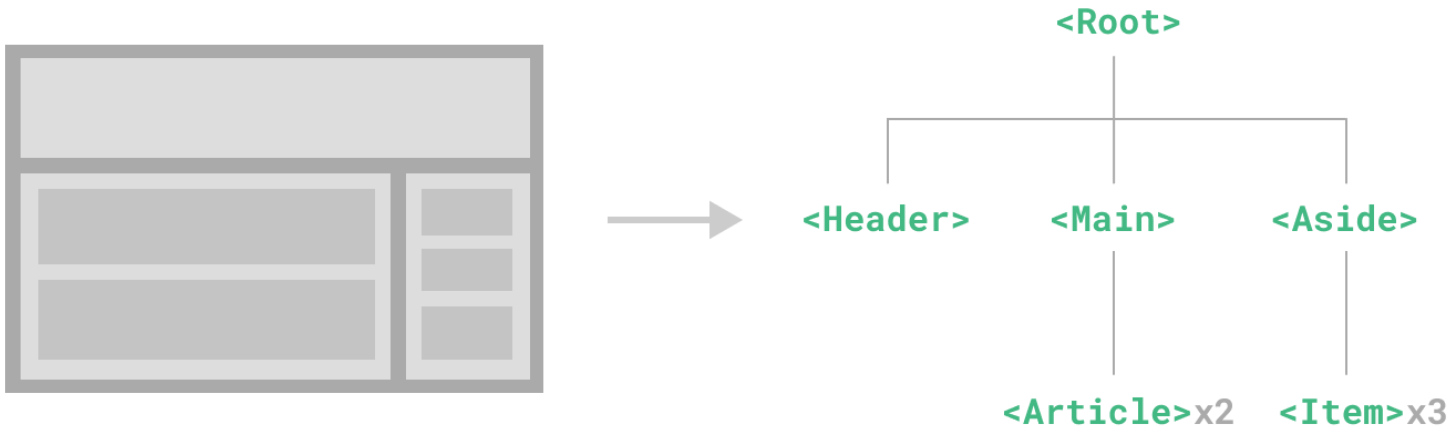
- Traque les dépendances et les effets de bord en une phase
- Il traque automatiquement chaque propriété réactive accédée durant son exécution synchrone

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Les composants

Principe des composants

- Les composants permettent de fractionner l'UI en morceaux indépendants et réutilisables
- Vue implémente son propre modèle de composant en encapsulant le contenu et la logique au sein de chaque composant



Définir un composant

Chaque composant est défini dans un fichier `.vue` contenant la logique ainsi que le template de ce composant

```
// ButtonCounter.vue
<script setup>
import { ref } from 'vue'

const count = ref(0)
</script>

<template>
  <button @click="count++">You clicked me {{ count }} times.</button>
</template>
```

Utiliser un composant

- Pour utiliser un composant enfant, il faut l'importer dans le parent
- Les composants sont noté en PascalCase pour les différencier des éléments HTML

```
<script setup>
import ButtonCounter from './ButtonCounter.vue'
</script>

<template>
  <h1>Here is a child component!</h1>
  <ButtonCounter />
</template>
```

Les props

- Une prop est une propriété personnalisée que l'on peut passer à un composant
- Les props sont utilisées pour transmettre des données d'un composant parent à un composant enfant

```
<script setup>
defineProps(['title'])
</script>

<template>
  <h4>{{ title }}</h4>
</template>
```

```
<BlogPost title="My journey with Vue" />
<BlogPost title="Blogging with Vue" />
<BlogPost title="Why Vue is so fun" />
```

defineProps

- `defineProps` est une macro de compilation qui est seulement accessible à l'intérieur de `<script setup>` et ne nécessite pas d'être explicitement importée
- Les props déclarées sont automatiquement exposées au template
- `defineProps` retourne également un objet contenant toutes les propriétés passées au composant, de manière à ce que l'on puisse y accéder en JavaScript si nécessaire

defineProps avec un tableau

- `defineProps()` permet de déclarer des propriétés avec un tableau de chaînes de caractères
- Cette méthode est pratique pour déclarer des propriétés rapidement sans ajouter de vérifications sur les props passées au composant

defineProps avec un objet

- Il est possible de passer des props à l'aide d'un objet
- Chaque propriété de l'objet sert de déclaration, la clé est le nom du prop et sa valeur le type attendu

```
defineProps({
  title: String,
  likes: Number,
});
```

Convention de nommage

Utilisation du **camelCase** pour la définition de props

```
defineProps({
  greetingMessage: String,
});
```

Utilisation du **kebab-case** pour la transmission de props

```
<MyComponent greeting-message="hello" />
```

Props dynamique et statique

- Utilisation de prop statique:

```
<BlogPost title="My journey with Vue" />
```

- Les props dynamique utilise la syntaxe de v-bind:

```
<BlogPost :title="post.title" />
```

Lier l'ensemble des propriétés d'un objet

v-bind sans argument permet de lier toutes les propriétés d'un objet au composant :

```
const post = {
  id: 1,
  title: "My Journey with Vue",
};
```

```
<BlogPost v-bind="post" />
```

Props d'objet

- Ici `v-bind` est utilisé pour passer des props dynamiques
-

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

Flux de données à sens unique

- Les props passés dans un composant enfant sont en lecture seule
- chaque fois que le composant parent est mise à jour, les props des enfants sont actualisées

```
const props = defineProps(["foo"]);

// ❌ avertissement, les props sont en lecture seule !
props.foo = "bar";
```

Validation de props

Les composants peuvent spécifier des exigences pour leurs props :

- `required` : true/false
- `default` :
  - pour un type primitif, une valeur par défaut: 0, "", false, ...
  - pour un objet ou un tableau il faut utiliser une fonction factory
  - Pour une validation personnalisée il est possible d'utiliser la fonction `validator()`

Exemple de validation

```
  propB: [String, Number],
  propC: {
    type: String,
    required: true
  },
  propD: {
    type: Number,
    default: 100
  },
```

```
  propE: {
    type: Object,
    default(rawProps) {
      return { message: 'hello' }
    }
  },
  propF: {
    validator(value) {
      return ['success', 'warning', 'danger'].includes(value)
    }
  },
```

Enregistrement global

Pour rendre les composants disponibles partout dans l'application vue on peut l'enregistrer avec la méthode component

```
app
  .component("ComponentA", ComponentA)
  .component("ComponentB", ComponentB)
  .component("ComponentC", ComponentC);
```

Pour l'utiliser dans un template :

```
<ComponentA />
```

Il est recommandé d'utiliser le PascalCase pour nommer les composants

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Événements de composant

Émettre un évènement

Un composant peut émettre des événements personnalisés directement à partir du template à l'aide de la méthode native `$emit` :

```
<button @click="$emit('someEvent')">click me</button>
```

Le composant peut l'écouter avec `v-on` :

```
<MyComponent @some-event="callback" />
```

Il est recommandé d'utiliser le kebab-case pour les écouteurs d'évents

Argument d'évènement

Il est parfois utile d'émettre une valeur spécifique avec un événement.

Pour cela il suffit de l'ajouter en second argument à `$emit()` :

Dans le composant:

```
<button @click="$emit('increaseBy', 1)">
  Increase by 1
</button>
```

Dans le parent:

```
<MyButton @increase-by="(n) => count += n" />
```

Déclarer les événements émis

Dans `<script setup>` :

```
import { defineEmits } from "vue";
const emit = defineEmits(["inFocus", "submit"]);
const buttonClick = () => emit("submit");
```

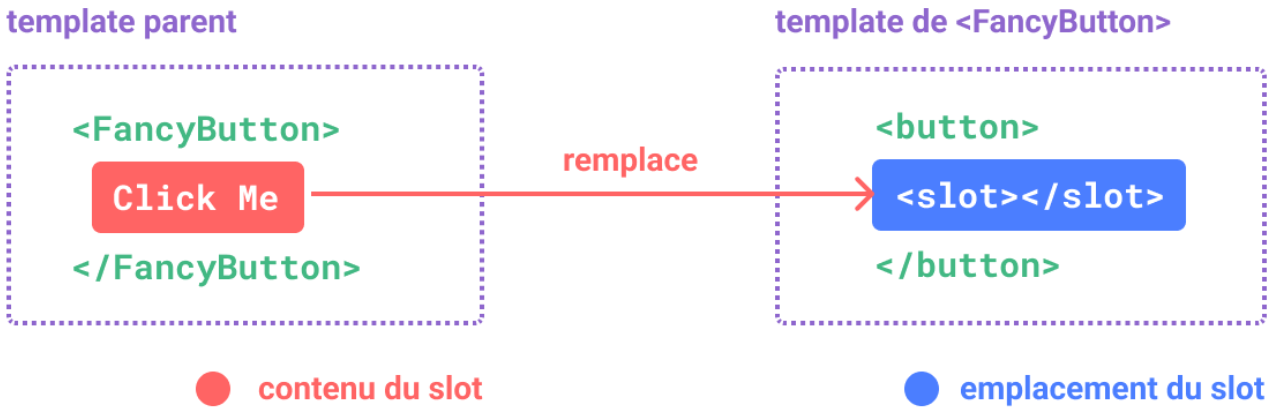
```
<template>
  <div>
    <button @click="buttonClick">Cliquez ici</button>
  </div>
</template>
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Slots

Qu'est-ce qu'un slot

L'élément `<slot>` est un emplacement qui indique où le contenu fourni par le parent doit être affiché.
L'utilisation de slots rend les composants plus facilement réutilisables.



valeur par défaut

Vue nous donne la possibilité de définir une valeur par défaut dans un slot.

Si une valeur est passée dans le composant parent, celle-ci est remplacée.

```
<button type="submit">
  <slot>
    Submit <!-- contenu par défaut -->
  </slot>
</button>
```

Slots nommés

Il est parfois utile d'avoir plusieurs slots dans un composant, cela est possible grâce aux slots nommés.

`v-slot` possède également un raccourci: `#`

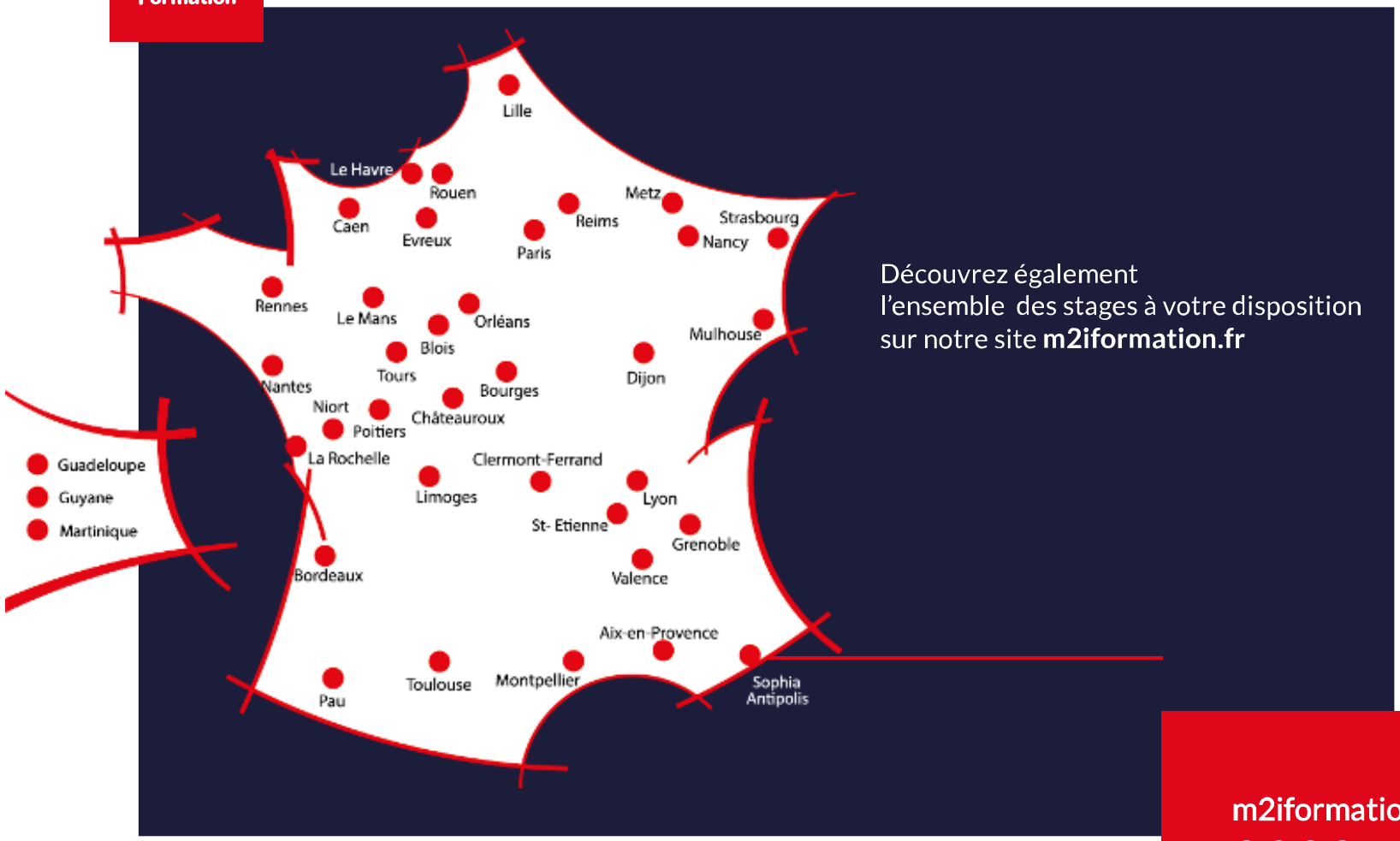
```
<header>
  <slot name="header"></slot>
</header>
<main>
  <slot></slot>
</main>
<footer>
  <slot name="footer"></slot>
</footer>
```

```
<BaseLayout>
  <template v-slot:header>
    <!-- contenu pour le slot header -->
  </template>
    <template #default>
    <p>A paragraph for the main content.</p>
  </template>
  <template #footer>
    <p>Here's some contact info</p>
  </template>
</BaseLayout>
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Merci pour votre attention

Des questions ?



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr