Principes et Commandes VueJS

VueJS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbole | Signification | Code | | Commentaires |
|  | Principes généraux et syntaxe |  | |  |
| Component.vue | Composant VueJS basique |  |  |  |
| template | un composant VueJS est constitué d'un seul fichier monComposant.vue contenant 3 éléments :   * template (html) * script (TS, JS) * style (CSS, SCSS) | <template>  […]  </template>  <script setup lang="ts">  import { ref } from 'vue'  […]  </script>  <style lang="scss">  […]  </style> |  |  |
|  | Composant Parent/Enfant |  |  |  |
| <monCompEnfant /> | import de composant enfant dans un composent parent | *Parent.vue*  <template>  <ChildComp msg="Valeur en dur" :count="monCompteur" />  </template>  <script setup>  import ChildComp from './ChildComp.vue'  const monCompteur = ref(0)  </script> | *ChildComp.vue*  <template>  <h1>Mon titre</h1>  <p>{{msg || 'Pas de message'}}<p>  <p>Valeur = {{count || '0'}}</p>  </template>  <script setup>  const props = **defineProps({**  **msg**: String,  **count**: Number,  monObjet: Object as PropType<MonInterface>  **})**  </script> |  |
| defineProps({ }) | macro de compilation perméttant de définir des propriétés qui seront alimentées par le composant parent |  |
| defineEmits([ ])  emit( , ) | émission d'un évenement depuis le composant enfant vers le composant parent | *Parent.vue*  <template>  <ChildComp @response="handleResponse" />  </template>  <script setup>  import ChildComp from './ChildComp.vue'  const monMessage = ref('')  handleResponse(msgEnfant) {  monMessage.value = msgEnfant  }  </script> | *ChildComp.vue*  <template>  <h1>Mon titre</h1>  </template>  <script setup>  const emit = **defineEmits([**'**response**'**])**  **emit(**'**response**'**,** 'Mon message envoyé par enfant'**)**  </script> |  |
| <slot> | la balise slot permet à un composant enfant d'afficher une partie du template transmis par le parent  elle permet aussi de définir à l'interieur de la balise un contenu par défaut que l'enfant affichera tant que le parent ne lui a pas passé d'éléments du template | *Parent.vue*  <template>  <ChildComp>  **<p>Parti de template transmis à l'enfant</p>**  </ChildComp>  </template>  <script setup>  import ChildComp from './ChildComp.vue'  </script> | *ChildComp.vue*  <template>  <h1>Mon titre</h1>  <slot>  **<div>Pas de slot transmis par le parent</div>**  </slot>  </template>  <script setup>  </script> |  |
|  |  |  |  |  |
| Réactivité | si changement de valeur dans cette objet coté Script => actualisation de la valeur dans le HTML | | | |
|  | **déclaration d'une variable réactive** |  |  |  |
| reactive( ) | rend un objet réactif  Rq: *ne prend en param que des objets* | const counter = reactive({  count: 0  }) |  |  |
| ref( ) | rend un élément réactif et créé un objet qui expose la valeur interne sous la propriété .value  Rq: *prend en param tout type* | const opened = ref(true)  function close(): void {  opened.value = false  } | <div v-if="opened">  […]  </div> |  |
| computed(() => { }) | créé une référence caclulée qui utilise plusieurs références réactive  recalcule et mets à jour le DOM si l'une de ces référence change | const **hideCompleted** = ref(false)  const **todos** = ref([  { id: id++, text: 'Apprendre le HTML', done: true },  { id: id++, text: 'Apprendre Vue', done: false }  ])  const **filteredTodos** = computed(() => {  if(**hideCompleted.value**) {  return **todos.value**.filter(l => !l.done)  }  return **todos.value**  }) | ~~<li v-for="todo in todos">~~  <li v-for="todo in **filteredTodos**"> | Rq: exemple complet => voir directive conditionnelle v-for |
| watch( , ) | observateur perméttant de mettre en place un effet de bord, cad un traitement anexe lorsque la valeur de l'élément réactif passé en paramètre change | const count = ref(0)  watch(count, maFonction) | avec fonction anonyme :  watch(count, (newCount) => {  //traitement  }) |  |
|  | directives de liaisons |  |  |  |
| v-bind:xx  :xx | directive permétant de synchroniser la valeur d'un élément réactif à l'un des atributs de balise du DOM  Rq: *la syntaxe* :xx *est un raccourci de* v-bind:xx | const color = ref('red') | <div v-bind:class="color"></div>  <style>  .red {  color: red;  }  </style> |  |
| v-on:xx  @xx | écoute les évenements en provenance du DOM | const count = ref(0)  function increment() {  count.value++  } | <button @click="increment**()**">{{ count }} </button>  Rq: *contrairement à Angular, les parenthèses de la ref à la fonction sont facultatives ici* |  |
| v-model | pour créer une liaison bidirectionnelle, il est possible d'utiliser :   * les 2 directives précédentes * ou la synthaxe raccourci v-model | *liaison bidirectionnelle avec les 2 directives :*  const text = ref('')  function onInput(e) {  text.value = e.target.value  }  *liaison bidirectionnelle avec v-model :*  const text = ref('') | <input :value="text" @input="onInput" >  <p>{{text}}</p>  <input v-model="text" >  <p>{{text}}</p> |  |
|  |  |  |  |  |
|  | Cycle de vie d'un component |  |  |  |
| ref=" "  ref(null) | création d'une référence à un élément du DOM afin de personnaliser la réactivité | const **divEltRef** = ref(null)  onMounted(() => {  **divEltRef**.value.textContent = "Valeur modifiée"  }) | <div ref="**divEltRef**">  […]  </div> |  |
| onBeforeMount(() => { }) | hooks du cycle de vie du composant (non exhaustif) |  |
| onMounted(() => { }) |  |
| onBeforeUpdate(() => { }) |  |
| onUptated(() => { }) |  |
| onBeforeUnmount(() => { }) |  |
| onUnmounted(() => { }) |  |
|  |  |  |  |  |
|  | dirrectives conditionnelles |  |  |  |
| v-if  v-else | créé un élément du DOM de manière conditionnelle | const opened = ref(true)  function close(): void {  opened.value = false  } | <div v-if="opened">  […]  </div>  <div v-else>Ma div alternative</div> |  |
| v-for | boucle | let id = 0  const newTodo = ref('')  const hideCompleted = ref(false)  const todos = ref([  { id: id++, text: 'Apprendre le HTML', done: true },  { id: id++, text: 'Apprendre le JavaScript', done: true },  { id: id++, text: 'Apprendre Vue', done: false }  ])  const filteredTodos = computed(() => {  if(hideCompleted.value) {  return todos.value.filter(l => !l.done)  }  return todos.value  })  function addTodo() {  todos.value.push({ id: id++, text: newTodo.value, done: false })  newTodo.value = ''  }  function removeTodo(todo) {  todos.value = todos.value.filter((t) => t !== todo)  } | <template>  <form @submit.prevent="addTodo">  <input v-model="newTodo" required placeholder="Nouvelle tâche">  <button>Ajouter une tâche</button>  </form>    <ul>  <li v-for="todo in filteredTodos" :key="todo.id">  <li v-for="(todo, index) in filteredTodos" :key="todo.id">  <input type="checkbox" v-model="todo.done">  <span :class="{ done: todo.done }">{{ todo.text }}</span>  <button @click="removeTodo(todo)">X</button> </li>  </ul>  <button @click="hideCompleted = !hideCompleted">  {{ hideCompleted ? 'Afficher toutes' : 'Cacher complétées' }}  </button>  </template>  <style>  .done {  text-decoration: line-through;  }  </style> |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  | |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  | |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |