RÉFÉRENCE RAPIDE VUE.JS

Templates, réactivité, composants, API de composition, routeur

Syntaxe des templates

Texte et expressions

```
<span>{{ message }}</span>
<span>{{ count + 1 }}</span>
<span>{{ ok ? 'Yes' : 'No' }}</span>
<span v-html="rawHtml"></span>
```

Directives

`{{ expr }}`	Interpolation de texte
`v-bind:attr / :attr`	Lier un attribut à une expression
`v-on:event / @event`	Attacher un écouteur d'événement
`v-model`	Liaison bidirectionnelle (formulaires)
`v-if / v-else-if / v-else`	Rendu conditionnel
`v-show`	Basculer le CSS display (reste dans le DOM)
`v-for`	Rendu de liste
`v-slot / #name`	Contenu d'un slot nommé

Liaison d'attributs

```
<img :src="imageUrl" :alt="desc" />
<div :class="{ active: isActive }"></div>
<div :style="{ color: textColor }"></div>
<button :disabled="isLoading">Submit</button>
```

Réactivité

ref (primitifs)

```
import { ref } from 'vue'

const count = ref(0)
console.log(count.value)  // 0
count.value++             // reactive update
```

reactive (objets)

```
import { reactive } from 'vue'

const state = reactive({ count: 0, name: 'Vue' })
state.count++  // no .value needed
```

ref vs reactive

`ref()`	N'importe quel type ; accès via `.value` dans le script
`reactive()`	Objets/tableaux uniquement ; accès direct aux propriétés
`Template`	Les deux se déballent automatiquement (`.value` inutile)
`Destructure`	`reactive` perd la réactivité ; utiliser `toRefs()`

Propriétés calculées et observateurs

Propriétés calculées

```
import { ref, computed } from 'vue'

const items = ref([1, 2, 3, 4, 5])
const evenItems = computed(() =>
  items.value.filter(n => n % 2 === 0)
)
```

Les valeurs calculées sont mises en cache et ne se réévaluent que si les dépendances changent

Observateurs

```
import { ref, watch, watchEffect } from 'vue'

const query = ref('')

// Watch specific source
watch(query, (newVal, oldVal) => {
  console.log(`Changed: ${oldVal} → ${newVal}`)
})

// Auto-track dependencies
watchEffect(() => {
  console.log(`Query is: ${query.value}`)
})
```

Options d'observateur

`immediate: true`	Exécuter le rappel immédiatement à la création
`deep: true`	Observer en profondeur les objets imbriqués
`flush: 'post'`	Exécuter après la mise à jour du DOM
`once: true`	Se déclencher une seule fois puis s'arrêter

Composants

Composant à fichier unique (SFC)

```
<script setup>
import { ref } from 'vue'
const count = ref(0)
</script>

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

<style scoped>
button { font-size: 1.2em; }
</style>
```

Enregistrer des composants

```
<!-- Auto-imported with <script setup> -->
<script setup>
import MyButton from './MyButton.vue'
</script>

<template>
  <MyButton label="Click me" />
</template>
```

Blocs SFC

`<script setup>`	API de composition (recommandée)
`<template>`	Template HTML
`<style scoped>`	CSS limité au composant
`<style module>`	CSS Modules (objet $style)

Props et événements

Définir des props

```
<script setup>
const props = defineProps({
  title: String,
  count: { type: Number, default: 0 },
  items: { type: Array, required: true }
})
</script>
```

Émettre des événements

```
<script setup>
const emit = defineEmits(['update', 'delete'])

function handleClick() {
  emit('update', { id: 1, value: 'new' })
}
</script>
```

Utilisation depuis le parent

```
<ChildComponent
  :title="pageTitle"
  :count="total"
  @update="handleUpdate"
  @delete="handleDelete"
/>
```

v-model sur les composants

```
<!-- Parent -->
<CustomInput v-model="search" />

<!-- CustomInput.vue -->
<script setup>
const model = defineModel()
</script>
<template>
  <input :value="model" @input="model = $event.target.value" />
</template>
```

Slots

Slot par défaut

```
<!-- Card.vue -->
<template>
  <div class="card">
    <slot>Fallback content</slot>
  </div>
</template>

<!-- Usage -->
<Card><p>Custom content here</p></Card>
```

Slots nommés

```
<!-- Layout.vue -->
<template>
  <header><slot name="header" /></header>
  <main><slot /></main>
  <footer><slot name="footer" /></footer>
</template>

<!-- Usage -->
<Layout>
  <template #header><h1>Title</h1></template>
  <p>Main content</p>
  <template #footer><span>Footer</span></template>
</Layout>
```

Slots délimités

```
<!-- List.vue -->
<ul>
  <li v-for="item in items" :key="item.id">
    <slot :item="item" />
  </li>
</ul>

<!-- Usage -->
<List :items="todos">
  <template #default="{ item }">
    <span>{{ item.text }}</span>
  </template>
</List>
```

API de composition

Fonction composable

```
// useMouse.js
import { ref, onMounted, onUnmounted } from 'vue'

export function useMouse() {
  const x = ref(0)
  const y = ref(0)
  function update(e) {
    x.value = e.pageX
    y.value = e.pageY
  }
  onMounted(() => window.addEventListener('mousemove', update))
  onUnmounted(() => window.removeEventListener('mousemove',
update))
  return { x, y }
}
```

Utiliser des composables

```
<script setup>
import { useMouse } from './useMouse'

const { x, y } = useMouse()
</script>
<template>
  <p>Mouse: {{ x }}, {{ y }}</p>
</template>
```

provide / inject

```
// Parent
import { provide, ref } from 'vue'
const theme = ref('dark')
provide('theme', theme)

// Descendant (any depth)
import { inject } from 'vue'
const theme = inject('theme', 'light') // default
```

Routeur (Vue Router)

Définitions de routes

```
import { createRouter, createWebHistory } from 'vue-router'

const router = createRouter({
  history: createWebHistory(),
  routes: [
    { path: '/', component: Home },
    { path: '/about', component: About },
    { path: '/user/:id', component: User },
  ]
})
```

Navigation dans le template

```
<router-link to="/">Home</router-link>
<router-link :to="{ name: 'user', params: { id: 1 } }">
  User 1
</router-link>
<router-view />
```

Navigation programmatique

```
import { useRouter, useRoute } from 'vue-router'

const router = useRouter()
const route = useRoute()

router.push('/about')
router.push({ name: 'user', params: { id: 1 } })
console.log(route.params.id)
```

Fonctionnalités de route

`/user/:id`	Segment dynamique (`route.params.id`)
`name: 'user'`	Route nommée pour la navigation programmatique
`children: [...]`	Routes imbriquées
`beforeEnter`	Garde de navigation par route
`meta: { auth: true }`	Métadonnées personnalisées pour les gardes
`redirect: '/new-path'`	Redirection de route

Hooks de cycle de vie

Ordre des hooks

`onBeforeMount`	Avant le rendu DOM initial
`onMounted`	DOM prêt (récupérer des données, ajouter des écouteurs)
`onBeforeUpdate`	Avant que l'état réactif ne re-rende le DOM
`onUpdated`	Après le re-rendu du DOM
`onBeforeUnmount`	Avant la destruction du composant
`onUnmounted`	Nettoyage (supprimer les écouteurs, minuteries)

Utilisation

```
<script setup>
import { onMounted, onUnmounted } from 'vue'

onMounted(() => {
  console.log('Component mounted')
})

onUnmounted(() => {
  console.log('Cleanup here')
})
</script>
```

Listes et conditions

v-for

```
<li v-for="item in items" :key="item.id">
  {{ item.name }}
</li>
<li v-for="(item, index) in items" :key="item.id">
  {{ index }}: {{ item.name }}
</li>
<div v-for="(val, key) in obj" :key="key">
  {{ key }}: {{ val }}
</div>
```

Toujours utiliser :key avec v-for pour des mises à jour DOM efficaces

v-if vs v-show

`v-if`	Rendu conditionnel (ajouter/supprimer du DOM)
`v-else-if`	Chaîne else-if
`v-else`	Branche de repli
`v-show`	Basculer `display: none` (reste dans le DOM)

Utiliser v-show pour les bascules fréquentes, v-if pour les changements rares

Exemple de conditions

```
<div v-if="status === 'loading'">Loading...</div>
<div v-else-if="status === 'error'">Error!</div>
<div v-else>{{ data }}</div>
```

Gestion des formulaires

Bases de v-model

```
<input v-model="text" />
<textarea v-model="message"></textarea>
<input type="checkbox" v-model="checked" />
<select v-model="selected">
  <option value="a">A</option>
  <option value="b">B</option>
</select>
```

Modificateurs de v-model

`v-model.lazy`	Synchroniser sur `change` au lieu de `input`
`v-model.number`	Conversion automatique en nombre
`v-model.trim`	Supprimer automatiquement les espaces en début/fin

Modificateurs d'événements

`@click.prevent`	Appeler `preventDefault()`
`@click.stop`	Appeler `stopPropagation()`
`@click.once`	Se déclencher au plus une fois
`@keyup.enter`	Uniquement sur la touche Entrée
`@submit.prevent`	Empêcher la soumission par défaut du formulaire