VUE.JS 快速参考

模板、响应式、组件、Composition API、路由

模板语法

文本与表达式

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

指令

`{{ expr }}`	文本插值
`v-bind:attr / :attr`	将属性绑定到表达式
`v-on:event / @event`	绑定事件监听器
`v-model`	双向绑定（表单）
`v-if / v-else-if / v-else`	条件渲染
`v-show`	切换 display CSS（保留在 DOM 中）
`v-for`	列表渲染
`v-slot / #name`	具名插槽内容

属性绑定

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

响应式

ref（原始值）

```
import { ref } from 'vue'

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

reactive（对象）

```
import { reactive } from 'vue'

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

ref vs reactive

`ref()`	支持任意类型；在 script 中通过 `.value` 访问
`reactive()`	仅支持对象/数组；直接访问属性
`Template`	两者在模板中均自动解包（无需 `.value`）
`Destructure`	`reactive` 解构后失去响应性；用 `toRefs()`

计算属性与侦听器

计算属性

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

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

计算属性会缓存，只在依赖变化时重新计算

侦听器

```
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}`)
})
```

watch 选项

`immediate: true`	创建时立即执行回调
`deep: true`	深度侦听嵌套对象
`flush: 'post'`	在 DOM 更新后执行
`once: true`	只触发一次后停止

组件

单文件组件（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>
```

注册组件

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

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

SFC 块

`<script setup>`	Composition API（推荐写法）
`<template>`	HTML 模板
`<style scoped>`	组件作用域 CSS
`<style module>`	CSS Modules（`$style` 对象）

Props 与事件

定义 Props

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

触发事件

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

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

父组件用法

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

组件上的 v-model

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

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

插槽

默认插槽

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

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

具名插槽

```
<!-- 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>
```

作用域插槽

```
<!-- 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>
```

Composition API

组合式函数

```
// 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 }
}
```

使用组合式函数

```
<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
```

路由（Vue Router）

路由定义

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

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

模板导航

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

编程式导航

```
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)
```

路由特性

`/user/:id`	动态路由段（`route.params.id`）
`name: 'user'`	命名路由，用于编程式导航
`children: [...]`	嵌套路由
`beforeEnter`	单路由导航守卫
`meta: { auth: true }`	守卫使用的自定义元数据
`redirect: '/new-path'`	路由重定向

生命周期钩子

钩子执行顺序

`onBeforeMount`	初始 DOM 渲染前
`onMounted`	DOM 就绪（获取数据、绑定事件）
`onBeforeUpdate`	响应式状态触发 DOM 更新前
`onUpdated`	DOM 重新渲染后
`onBeforeUnmount`	组件销毁前
`onUnmounted`	清理（移除事件监听、定时器）

用法

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

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

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

列表与条件

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>
```

v-for 必须使用 :key 以确保高效的 DOM 更新

v-if vs v-show

`v-if`	条件渲染（从 DOM 中添加/移除）
`v-else-if`	else-if 链
`v-else`	默认分支
`v-show`	切换 `display: none`（保留在 DOM 中）

频繁切换用 v-show，少见变化用 v-if

条件示例

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

表单处理

v-model 基础

```
<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>
```

v-model 修饰符

`v-model.lazy`	在 `change` 事件时同步（而非 `input`）
`v-model.number`	自动转换为数字类型
`v-model.trim`	自动去除首尾空白

事件修饰符

`@click.prevent`	调用 `preventDefault()`
`@click.stop`	调用 `stopPropagation()`
`@click.once`	最多触发一次
`@keyup.enter`	仅在按下 Enter 键时触发
`@submit.prevent`	阻止表单默认提交行为