用表格输出，看上去不会太乱

# Vue 核心语法速查表

## 1. 响应式数据系统

| 特性 | 语法示例 | 说明 |
| --- | --- | --- |
| 数据声明 | `data() { return { count: 0 } }` | 组件状态初始化 |
| 双向绑定 | `<input v-model="message">` | 表单输入实时同步数据 |
| 响应式原理(Vue2) | `Object.defineProperty()` | 基于属性劫持的响应式实现 |
| 响应式原理(Vue3) | `Proxy` | 基于代理的深层响应式追踪 |

## 2. 模板指令系统

| 指令 | 语法示例 | 说明 |
| --- | --- | --- |
| 文本插值 | `{{ dynamicText }}` | 绑定文本内容 |
| 原始HTML | `<div v-html="rawHtml"></div>` | 渲染原始HTML |
| 属性绑定 | `<img :src="imageUrl">` | 动态绑定元素属性 |
| 条件渲染(v-if) | `<div v-if="isVisible">内容</div>` | 条件性渲染元素 |
| 条件渲染(v-show) | `<p v-show="hasError">错误</p>` | 基于CSS显示/隐藏元素 |
| 循环渲染 | `<li v-for="(item,index) in items">` | 列表渲染(必须配合 `:key` ) |
| 事件绑定 | `<button @click="handleClick">提交</button>` | 监听DOM事件 |
| 事件修饰符 | `<form @submit.prevent="onSubmit">` | 阻止默认行为 |
| 按键修饰符 | `<input @keyup.enter="submit">` | 特定按键触发 |

## 3. 计算与侦听

| 特性 | 语法示例 | 说明 |
| --- | --- | --- |
| 计算属性 | `computed: { fullName() { ... } }` | 基于依赖缓存的派生数据 |
| 侦听器 | `watch: { value(newVal) { ... } }` | 响应数据变化执行操作 |
| 深度侦听 | `watch: { obj: { deep: true, ... } }` | 监听对象深层变化 |

**4. 组件系统**

| 特性 | 语法示例 | 说明 |
| --- | --- | --- |
| 组件定义 | `Vue.component('comp-name', { ... })` | 全局组件注册 |
| Props传递 | `<child :title="postTitle"></child>` | 父→子数据传递 |
| 事件发射 | `this.$emit('update', newValue)` | 子→父事件通信 |
| 默认插槽 | `<slot>后备内容</slot>` | 内容分发入口 |
| 具名插槽 | `<slot name="header"></slot>` | 多内容分发区域 |
| 作用域插槽 | `<slot :item="item"></slot>` | 子组件向插槽传递数据 |
| 动态组件 | `<component :is="currentComp"></component>` | 动态切换组件 |

**5. 生命周期(Vue2)**

| 钩子函数 | 触发时机 |
| --- | --- |
| `beforeCreate` | 实例初始化后，数据观测/事件配置前 |
| `created` | 实例创建完成，数据观测/事件配置完成 |
| `beforeMount` | DOM挂载开始前 |
| `mounted` | DOM挂载完成 |
| `beforeUpdate` | 数据更新时，DOM重新渲染前 |
| `updated` | 数据更新后，DOM重新渲染完成 |
| `beforeDestroy` | 实例销毁前 |
| `destroyed` | 实例销毁后 |

**6. Vue3 新特性**

| 特性 | 语法示例 | 说明 |
| --- | --- | --- |
| Composition API | `setup() { ... }` | 逻辑组织新范式 |

| 特性 | 语法示例 | 说明 |
| --- | --- | --- |
| ref响应式 | `const count = ref(0)` | 基础类型响应式包装 |
| reactive响应式 | `const state = reactive({...})` | 对象类型响应式包装 |
| 生命周期更新 | `onMounted(() => {...})` | 组合式API生命周期 |
| Teleport组件 | `<Teleport to="body">...</Teleport>` | 传送内容到指定DOM位置 |
| Suspense组件 | `<Suspense><template #default>...</template></Suspense>` | 异步组件加载状态处理 |

**7. 状态管理**

| 概念(Vuex) | 说明 |
| --- | --- |
| `state` | 存储应用状态数据 |
| `mutations` | 同步修改状态的方法 |
| `actions` | 异步操作和提交mutations |
| `getters` | 计算派生状态 |
| **概念(Pinia)** | |
| `store` | 状态存储单元 |
| `state` | 响应式状态 |
| `actions` | 操作状态的方法 |
| `getters` | 状态计算属性 |

**8. 路由管理**

| 特性 | 语法示例 | 说明 |
| --- | --- | --- |
| 路由声明 | `{ path: '/user/:id', component: User }` | 定义路由规则 |
| 路由链接 | `<router-link to="/home">首页</router-link>` | 声明式导航 |
| 路由视图 | `<router-view></router-view>` | 路由组件渲染出口 |
| 编程式导航 | `this.$router.push('/profile')` | JS控制路由跳转 |
| 路由守卫 | `beforeEach(to, from, next)` | 全局路由拦截逻辑 |

**设计原则**

| 原则 | 说明 | 体现形式 |
| --- | --- | --- |
| 声明式渲染 | 描述”应该是什么”而非”如何更新” | 模板语法、计算属性 |
| 组件驱动 | 高内聚、低耦合的UI单元 | 单文件组件(.vue) |
| 单向数据流 | 数据向下，事件向上 | Props向下传递，$emit向上通知 |
| 渐进式框架 | 按需引入功能模块 | 核心库→路由→状态管理→工具链 |
| 虚拟DOM | 高效渲染机制 | 差异比对最小化DOM操作 |

此表格全面涵盖了 Vue 的核心语法和设计理念，可作为日常开发的快速参考指南。Vue3 在保留核心思想的同时，通过 Composition API 提供了更灵活的逻辑组织方式，特别适合复杂应用的开发。