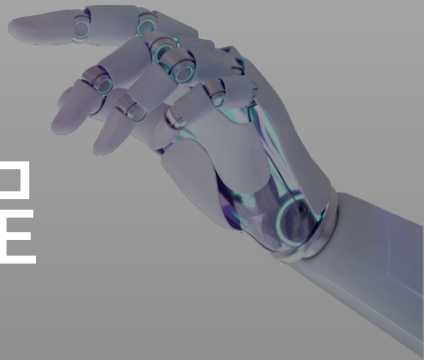


Web 前端编程

Web Front-end Programming

尹庆 Yin Qing

qyin@cuit.edu.cn





渐进式 JavaScript 框架

<https://cn.vuejs.org>

为什么使用Vue

- 手动操作 DOM 难以实现响应式数据管理：实时更新显示后端提供的**动态数据**

```
1  <body>
2    <div id="box">动态数据</div>
3
4    <script>
5      let value = '这是内容'
6      const box = document.querySelector('#box')
7      box.innerHTML = value
8
9      value = '这是修改后的内容'
10     box.innerHTML = value
11
12     // ... 继续修改
13   </script>
14 </body>
```

为什么使用Vue

- Vue（发音/vju:/, 类似view）：用于构建用户界面的渐进式 JavaScript 框架
 - 渐进式：一个项目可以逐步集成 Vue，而不需要一开始就All in Vue

开发优点：



- 简化DOM操作的代码编写
- 响应式数据处理：数据变化，页面视图自动更新
- 将数据和界面分离，降低代码耦合度

1 Vue 使用基本步骤 – 导入Vue

- 全局引入(`src`)： `Vue` 被定义为一个全局变量（不推荐）
- 模块引入(`type="module"`)： 只引入需要的 `Vue` 组件（推荐）

```
1  <!-- 全局引入 -->
2  <script src="./vue.global.js"></script>
3
4  <!-- 模块引入 -->
5  <script type="module">
6    import { createApp } from './vue.esm-browser.js'
7  </script>
```

注意

-  开发版本：包含完整的警告和调试模式， `vue.esm-browser.js`
-  生产版本：删除了警告，压缩文件， `vue.esm-browser.prod.js`

1 Vue 使用基本步骤

```
1  <!-- 1. 准备 HTML 容器：用于呈现内容 -->
2  <div id="app"> {{message}} </div>
3  <script type="module">
4      // 2. 导入Vue开发包
5      import { createApp } from './vue.esm-browser.js'
6      // 3. 创建应用实例对象
7      const app = createApp({
8          // 4. 提供数据：用于在 HTML 容器中显示
9          data(){
10              return {
11                  message: 'Hello World!',
12              }
13          }
14      })
15      // 5. 指定挂载点：选中准备的 HTML 容器，类似`querySelector`
16      app.mount('#app')
17  </script>
```

1 创建 Vue 应用示例

- 使用 `createApp` 函数创建Vue应用示例（对象）

```
1  <script type="module">
2      import { createApp } from './vue.esm-browser.js'
3      const app = createApp()
4      console.log(typeof(app))
5  </script>
```

2 指定挂载点

- Vue实例`.mount()` 方法：类似 `querySelector` ，用于选中指定的HTML容器
 - 参数：可以是一个实际的 DOM 元素或是一个 CSS 选择器字符串

示例

```
1 <!-- HTML 代码 -->
2 <div id="app"></div>
```

```
1 // JavaScript 代码
2 app.mount('#app')
```


补充：JS 简写语法

- ES6 引入了一种“用于把方法名直接赋给函数”的简写语法
 - 省略了 `function` 关键字和属性冒号：

```
1 // 常规写法
2 const obj = {
3   greet: function() {
4     return 'Hello!';
5   }
6 };
```

```
1 // 简写方式
2 const obj2 = {
3   greet() {
4     return 'Hello!';
5   }
6 };
```

3 提供显示数据

- `data` 方法：提供数据，用于在 HTML 容器中显示

```
1  const app = createApp({
2    data() {
3      return {
4        message: 'Hello World!'
5      }
6    }
7  })
8  app.mount('#app')
```

🤔 问题：如何让 `message` 数据显示在 HTML 容器里？

4 插值表达式

- `{{表达式}}`：用于在 HTML 容器中显示绑定到 Vue 实例中数据值

```
1 <div id="app"> {{message}} </div>
```

- `data` 方法+ 插值表达式：实现数据的响应式处理


5 声明方法

- `methods` 属性：为Vue 对象实例（组件）添加方法

```
1  const app = createApp({
2    methods: {
3      print () {
4        console.log('Hello World!')
5      }
6    }
7  })
8
9  app.mount('#app')
```

```
1  <!-- HTML中调用 -->
2  <div id="app">
3    {{ print() }}
4  </div>
```

6 Vue 开发者工具

- Vue 浏览器插件 Vue Devtools 下载地址：
 - Chrome 官方商店
 - Chrome 非官方商店（极简插件）
- F12 调试窗口菜单栏 > Vue

7 Vue 指令

- Vue 指令：带有 `v-` 前缀 的HTML元素属性，用于实现不同功能

- 15种内置指令

示例

```
1 <!-- `v-html="表达式"` 用于设置元素的 `innerHTML` -->
2 <div id="app">
3   <div v-html="message"></div>
4 </div>
```

```
1 data() {
2   return {
3     message: '<h1>Hello World!</h1>' }
4 }
```



相比直接在元素内容中使用插值表达式， `v-html` 能够解析标签

7 Vue 指令

- 控制元素显示与隐藏：
 - `v-show`：通过是否设置 CSS `display: none` 来控制显示/隐藏
 - `v-if`：通过创建/删除HTML元素来控制元素的显示/隐藏 (条件渲染)

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

💡 选择哪一个？

- `v-show`：需要频繁切换场景，如鼠标悬停购物车列表
- `v-if`：切换条件固定，不频繁切换的场景，如广告弹窗关闭

7 Vue 指令

- `v-if`, `v-else-if`, `v-else` 实质：调用对应 JS 的语法 + DOM 操作

```
1 <div id="app">
2   <p v-if="value > 10">条件值大于10</p>
3   <p v-else-if="value === 10">条件值等于10</p>
4   <p v-else>条件值小于10</p>
5 </div>
```

```
1 render() {
2   if (this.value > 10) {
3     return createElement('p', '条件值大于10');
4   } else if (this.value === 10) {
5     return createElement('p', '条件值等于10');
6   } else {
7     return createElement('p', '条件值小于10');
8   }
9 }
```


7 Vue 指令

- `v-for`：用于在模板中循环遍历数组或对象，**动态多次生成**对应的 DOM 元素
 - JavaScript `for` 循环的封装，可遍历数组、对象等

语法

```
1 <div v-for="(item, index) in items"></div>
2 <div v-for="(value, key) in object"></div>
3 <div v-for="(value, name, index) in object"></div>
```

示例

```
1 <ul id="app">
2   <li v-for="item in items">{{ item }}</li>
3 </ul>
```

- ⚠️💡 `v-for` 应用于需要重复渲染的元素（如 `li`），不应该放在结构容器（如 `ul`）

7 Vue 指令

- `v-model` : 用于表单输入元素上创建**双向**绑定
 - 双向绑定：一边变化，另一边自动跟着变化
 - 仅限： `<input>` , `<select>` , `<textarea>`

示例

```
1 <p>Message is: {{ message }}</p>
2 <input v-model="message" placeholder="edit me" />
```

7 Vue 指令

- `v-on:` 或 `@` : 给元素绑定事件监听器

语法

```
1 <button v-on:事件类型="要执行的函数名/行内语句"></button>
2 <button @事件类型="要执行的函数名/行内语句"></button>
```

💡 类似调用JS事件监听: 对象名.`addEventListener`('事件类型', 要执行的函数)

示例

```
1 <!-- 绑定method中的方法名 (函数) -->
2 <button v-on:click="doThis"></button>
3
4 <!-- 行内声明: 结合data内的属性名 -->
5 <button @click="count = count + 2"></button>
```

综合案例



任务清单(TodoList)

任务清单

请输入新任务

添加任务

☐ 练习篮球

☐ 练习铁山靠

☐ 练习RAP

单文件组件

- 组织项目代码的两种思路：

- 按类型分离：HTML、CSS、JS 分别放在不同的文件中
- 按功能分离：使用一个文件将HTML、CSSS、JS代码集中在一起

- 单文件组件 (Single-File Components, SFC)： 用于将一个组件的逻辑、模板和样式封装在同一文件里（文件后缀 `.vue`）

- 组件的逻辑(`<script>`) – JavaScript
- 组件的模板(`<template>`) – HTML
- 组件的样式(`<style>`) – CSS

单文件组件示例

```
1  <!-- App.vue -->
2  <script setup>
3  import { ref } from 'vue'
4  const count = ref(0)
5  </script>
6
7  <template>
8    <button @click="count++">Count is: {{ count }}</button>
9  </template>
10
11  <style scoped>
12    button {
13      font-weight: bold;
14    }
15  </style>
```

API 风格

- 选项式API(Options API): Vue2 风格, 推荐用于低复杂度的渐进增强应用场景
- 组合式API(Composition API): Vue3 风格, 推荐用于全Vue构建的复杂应用