讲师：李振良（阿良）

今天课题：《Vue前端开发》上篇

学院官网：www.aliangedu.cn

阿良个人微信

DevOps技术栈公众号

Vue前端开发（上篇）

- 认识Vue.js
- Vue常用指令
- Vue常用属性
- Vue常用指令之流程控制

认识Vue.js

- Vue介绍
- 引入Vue
- 声明式渲染
- 模板语法

Vue.js介绍

Vue.js（简称Vue） 是一套用于构建用户界面的渐进式前端框架。

Vue.js 核心实现：

- 响应式的数据绑定：当数据发生改变，视图可以自动更新，不用关心DOM操作，而专心数据操作。
- 可组合的视图组件：把视图按照功能切分成若干基本单元，可维护，可重用，可测试等特点。

官网：https://v3.cn.vuejs.org/

引入Vue.js

使用Vue的四种方式：

- 在HTML中以CDN包的形式导入
- 下载JS文件保存到本地再导入
- 使用npm安装
- 使用官方VueCli脚手架构建项目（不建议新手直接用）

参考文档：https://v3.cn.vuejs.org/guide/installation.html

Hello World示例

从一个Hello World例子开始:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>测试</title>
    <script src="https://unpkg.com/vue@next"></script>
</head>
<body>
    <div id="hello-vue">
      {{ message }}  <!--引用变量-->
    </div>
    <script type="text/javascript">
        const HelloVueApp = {
          data() {
            return {
              message: 'Hello Vue!!'  //变量名和值
            }
          }
        }
        Vue.createApp(HelloVueApp).mount('#hello-vue') //绑定元素
    </script>
</body>
</html>
```

声明式渲染

Vue.js 的核心是一个允许采用简洁的模板语法来声明式地将数据渲染进 DOM 的系统：

```
<div id="counter">
  Counter: {{ counter }}
</div>
<script type="text/javascript">
    const HelloVueApp = {
      data() {
        return {
          counter: 0
        }
      }
    }
    Vue.createApp(HelloVueApp).mount('#counter')
</script>
```

声明式渲染

现在数据和DOM已经被建立了关联，所有东西都是**响应式**的，可通过下面示例确认：

```
<div id="counter">
  Counter: {{ counter }}
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
      return {
        counter: 0
      }
    },
    mounted() {
    setInterval(() => {   // 周期性调用箭头函数
      this.counter++
    }, 1000)
    }
  }
  Vue.createApp(HelloVueApp).mount('#counter')
</script>
```

模板语法

Vue.js 使用了基于 HTML 的模板语法，允许开发者声明式地将 DOM 绑定至底层组件实例的数据。所有 Vue.js 的模板都是合法的 HTML，所以能被遵循规范的浏览器和 HTML 解析器解析。

数据绑定最常见的形式就是使用“双大括号” 语法在HTML中插入文本：

```
<span>Message: {{ msg }}</span>
```

{{msg}}将被替代对应组件实例中msg属性的值。无论何时，绑定的组件实例上msg属性发生改变，插值处内容都会更新。

Vue 常用指令

- 指令介绍
- v-text
- v-html
- v-bind
- v-on
- 指令缩写

指令介绍

指令：带有 v- 前缀的特殊属性。

指令的作用：当表达式的值改变时，将其产生的连带影响，响应式地作用于 DOM。

v-text

v-text作用与双大花括号作用一样，将数据填充到标签中。但没有闪烁问题！

```
<div id="hello-vue">
    <p v-text="msg"></p>
    <p>{{ msg }}</p>
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
     return {
        msg: "hello vue!",
     }
    }
  }
  Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

参考文档：https://v3.cn.vuejs.org/guide/template-syntax.html

v-html

某些情况下，从服务端请求的数据本身就是一个HTML代码，如果用双大括号会将数据解释为普通文本，而非HTML代码，为了输出真正的HTML，需要使用v-html指令：

```
<div id="hello-vue">
  {{ msg }}
  <span v-html="msg"></span>
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
      return {
        msg: "<span style='color: red'>Hello Vue!!</span>"
      }
    }
  }
  Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

v-on

在前端开发中，我们经常监听用户发生的事件，例如点击、拖拽、键盘事件等。

在Vue中如何监听事件呢？使用v-on指令

示例：监听按钮的点击事件

```
<div id="hello-vue">
    <p>点击次数: {{ counter }}</p>
    <button type="button" v-on:click="counter++">按钮</button>
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
     return {
        counter: 0,
     }
    }
  }
  Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

- v-on: 冒号后面是event参数，例如click、change

v-bind：介绍

v-bind：用于动态绑定一个或多个属性值，或者向另一个组件传递props值（这个后面再介绍）

应用场景：图片地址src、超链接href、动态绑定一些类、样式等等

v-bind：绑定超链接

示例：响应式地更新 HTML 属性

```
<div id="hello-vue">
  <a v-bind:href="url">阿良教育</a>
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
      return {
        url: "http://www.aliangedu.cn"
      }
    }
  }
  Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

- v-bind 指令后接收一个参数，以冒号分割。
- v-bind 指令将该元素的 href 属性与表达式 url 的值绑定。

v-bind：绑定Class

操作元素（标签）的 class 和 style 属性是数据绑定的一个常见需求。

例如希望动态切换class，为div显示不同背景颜色

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>测试</title>
    <script src="https://unpkg.com/vue@next"></script>
    <style>
        .test {
            width: 200px;
            height: 200px;
            background: grey;
        }
        .active {
            background: orange;
        }
    </style>
  </head>
  <body>
    <div id="hello-vue">
        <div v-bind:class="{active: isActive}" class="test"> <!--active这个class存在与否取决于数据属性isActive-->
        </div>
        <button type="button" @click="btn">增加样式</button>
    </div>
    <script type="text/javascript">
        const HelloVueApp = {
          data() {
            return {
                isActive: false,
            }
          },
          methods: {
              btn() {
                  // this.isActive = true;
                  // 实现动态切换
                  if (this.isActive) {
                      this.isActive = false
                  } else {
                      this.isActive = true
                  }
              }
          }
        }
        Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
  </body>
</html>
```

示例：给已有的div动态绑定一个class

参考文档：https://v3.cn.vuejs.org/guide/class-and-style.html

v-bind：绑定Style

示例：给已有的div动态绑定一个style

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>测试</title>
    <script src="https://unpkg.com/vue@next"></script>
    <style>
        .test {
            width: 200px;
            height: 200px;
            background: grey;
        }
    </style>
  </head>
  <body>
    <div id="hello-vue">
        <div v-bind:style="{background: background, fontSize: fontSize  + 'px'}" class="test">
            hello vue!
        </div>
    </div>
    <script type="text/javascript">
        const HelloVueApp = {
          data() {
            return {
                background: 'orange',
                fontSize: '24'
            }
          }
        }
        Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
  </body>
</html>
```

v-bind:style 的对象语法看着非常像 CSS，但其实是一个 JavaScript 对象。CSS 属性名可以用驼峰式 (camelCase) 或短横线分隔 (kebab-case，记得用引号括起来) 来命名。

v-bind：绑定Style

直接绑定到一个样式对象通常更好，这会让模板更清晰：

```
<div id="hello-vue">
    <div v-bind:style="styleObject" class="test">
        hello vue!
    </div>
</div>
<script type="text/javascript">
    const HelloVueApp = {
      data() {
        return {
            styleObject: {
                background: 'orange',
                fontSize: '24px'
            }
        }
      }
    }
    Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

指令缩写

v- 前缀作为一种视觉提示，用来识别模板中 Vue 特定的 属性。

但对于一些频繁用到的指令来说，就会感到使用繁琐。

因此，Vue 为 v-bind 和 v-on 这两个最常用的指令，提供了特定简写：

v-bind缩写

```
<!-- 完整语法 -->
<a v-bind:href="url"> ... </a>

<!-- 缩写 -->
<a :href="url"> ... </a>

<!-- 动态参数的缩写 -->
<a :[key]="url"> ... </a>
```

v-on缩写

```
<!-- 完整语法 -->
<a v-on:click="doSomething"> ... </a>

<!-- 缩写 -->
<a @click="doSomething"> ... </a>

<!-- 动态参数的缩写 -->
<a @[event]="doSomething"> ... </a>
```

Vue 常用属性

- 数据属性
- 方法
- 计算属性
- 监听属性

数据属性

组件的 data 选项是一个函数。Vue 会在创建新组件实例的过程中调用此函数。它应该返回一个对象，然后 Vue 会通过响应性系统将其包裹起来，并以 $data 的形式存储在组件实例中。为方便起见，该对象的任何顶级“属性”也会直接通过组件实例暴露出来：

```
<div id="hello-vue">
  <p>{{ msg }}</p>
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
      return {
        msg: "Hello Vue!"
      }
    }
  }
  const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
  console.log(vm.$data.msg) // => "Hello Vue!"
  console.log(vm.msg)       // => "Hello Vue!"

  // 修改vm.count 的值也会更新 $data.count
  vm.msg = "Hello Go!"
  console.log(vm.$data.msg) // => "Hello Go!"

  // 反之亦然
  vm.$data.msg = "Hello K8s!"
  console.log(vm.msg)  // => "Hello K8s!"
</script>
```

相等

```
const HelloVueApp = Vue.createApp ({
  data() {
    return {
      msg: "Hello Vue!"
    }
  }
})
const vm = HelloVueApp.mount('#hello-vue')
```

参考文档：https://v3.cn.vuejs.org/guide/data-methods.html#data-property

方法

方法（methods）：处理数据的函数。在methods选项中定义的函数称为方法。

示例：添加方法及调用

```
<div id="hello-vue">
  <p>{{ count }}</p>
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
      return {
        count: 4
      }
    },
    methods: {
        increment() {
            this.count++   // `this` 指向该组件实例
        }
    }
  }
  const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
  console.log(vm.count)       // => 4
  vm.increment()  // 调用方法
  console.log(vm.count)  // => 5
</script>
```

方法

在methods选项中定义的方法与data选项中的数据一样，可以在组件的模板中使用。

在模板中，它们通常被当做事件监听使用：

```
<div id="hello-vue">
    <p>{{ count }}</p>
    <button type="button" @click="increment">按钮</button>
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
      return {
        count: 4
      }
    },
    methods: {
        increment() {
            this.count++   // `this` 指向该组件实例，可以通过它访问data中任意数据
        }
    }
  }
  Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

计算属性

计算属性（computed）：根据所依赖的数据动态显示新的计算结果。

示例：需要在{{}}里添加计算再展示数据，例如统计分数

```
<div id="hello-vue">
  <span>总分：{{ math + language + english }}</span> <!--支持JS表达式-->
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
     return {
        math: 90,
        language: 88,
        english: 62
     }
    }
  }
  Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

计算属性

使用computed：

```
<div id="hello-vue">
  <span>总分：{{ sum }}</span> <!--支持简单的JS表达式-->
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
     return {
        math: 90,
        language: 88,
        english: 62
     }
    },
    computed: {
       sum: function() {
          return this.math + this.language + this.english;
       }
    }
  }
  Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

小结：计算属性一般就是用来通过其他的数据算出一个新数据，而且它有一个好处就是，它把新的数据缓存下来了，当其他的依赖数据没有发生改变，它调用的是缓存的数据，这就极大的提高了我们程序的性能。而如果写在methods里，数据根本没有缓存的概念，所以每次都会重新计算。这也是为什么不用methods的原因！

监听属性

监听属性（watch）：是一个观察动作，监听data数据变化后触发对应函数，函数有newValue（变化之后结果）和oldValue（变化之前结果）两个参数。

当需要在数据变化时执行异步或开销较大的操作时，这个方式是最有用的。

示例：监听变化

```
<div id="hello-vue">
  <p>消息：{{ msg }}</p>
  <p>观察的消息：{{ watchMsg }}</p>
  <button type="button" @click="btn">按钮</button>
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
     return {
        msg: 'hello vue!',
        watchMsg: ''
     }
    },
    methods: {
       // 点击按钮执行该函数，msg 重新赋新值，模拟 msg 值改变
       btn() {
          this.msg = 'hello go!';
       }
    },
    watch: {
       // 每当 msg 值发生变化时，执行该函数
       msg(newValue, oldValue) {
          console.log(newValue, oldValue)
          this.watchMsg = newValue;  // watchMsg 重新赋值，渲染到页面
       }
    }
  }
  Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

Vue常用指令之流程控制

- v-if
- v-show
- v-for

v-if、v-else、v-else-if

示例：判断一个元素是否显示

```
<div id="hello-vue">
    <p v-if="seen">现在你看到我了</p>
</div>
<script type="text/javascript">
    const HelloVueApp = {
      data() {
        return {
          seen: true
        }
      }
    }
    Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

示例：添加一个else块

```
<div id="hello-vue">
    <p v-if="seen">现在你看到我了</p>
    <p v-else>看不到我了</p>
</div>
<script type="text/javascript">
    const HelloVueApp = {
      data() {
        return {
          seen: false
        }
      }
    }
    Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

这里，v-if 指令将根据表达式 seen 的值的真假来插入/移除 <p> 元素。

v-if、v-else、v-else-if

v-if指令必须将它添加到一个元素上。如果想切换多个元素呢？　　示例：添加一个else块

此时可以把一个<template>元素当做不可见的包裹元素，并在上面使用v-if。最终的渲染结果将不包含<template>元素。

```
<div id="hello-vue">
    <template v-if="seen">
        <h1>标题</h1>
        <p>这是第一个段落。</p>
        <p>这是第二个段落。</p>
    </template>
</div>
<script type="text/javascript">
    const HelloVueApp = {
      data() {
        return {
          seen: true
        }
      }
    }
    Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

v-if、v-else、v-else-if

v-else-fi多分支：

```
<div id="hello-vue">
    <div v-if="type === 'A'">
        <p>A</p>
    </div>
    <div v-else-if="type === 'B'">
        <p>B</p>
    </div>
    <div v-else>
        <p>不是A和B! </p>
    </div>
</div>
<script type="text/javascript">
    const HelloVueApp = {
      data() {
        return {
          type: 'B'
        }
      }
    }
    Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

v-show

v-show：另一个用于条件性展示元素的指令，与v-if不同的是，v-show的元素始终会被渲染并保留再DOM中，所以v-show只是简单地切换元素的display CSS属性。

```
<div id="hello-vue">
  <p v-show="seen">现在你看到我了</p>
</div>
<script type="text/javascript">
  const HelloVueApp = {
    data() {
    return {
      seen: false
    }
    }
  }
  Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

v-for

可以用 v-for 指令基于一个数组来渲染一个列表。v-for 指令需要使用 item in items 形式的特殊语法，其中 items 是源数据数组，而 item 则是被迭代的数组元素的别名。

```
<div id="hello-vue">
  <ul>
    <li v-for="(c, i) in myArray">
      {{ i }}-{{ c }}
    </li>
  </ul>
</div>
<script type="text/javascript">
    const HelloVueApp = {
      data() {
        return {
            myArray: [
              '主机',
              '显示器',
              '键盘'
            ]
        }
      }
    }
    Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

示例：遍历数组

```
<div id="hello-vue">
  <ul>
    <li v-for="(v, k) in myObject">
      {{ k }}-{{ v }}
    </li>
  </ul>
</div>
<script type="text/javascript">
    const HelloVueApp = {
      data() {
        return {
            myObject: {
                host: '主机',
                displayer: '显示器',
                keyboard: '键盘'
            }
        }
      }
    }
    Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

示例：遍历对象

v-for：维护状态

当 Vue 正在更新使用 v-for 渲染的元素列表时，它默认使用“就地更新”的策略。如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序，而是就地更新每个元素，并且确保它们在每个索引位置正确渲染。

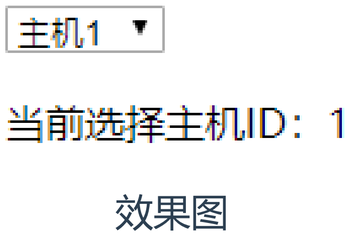
为了给 Vue 一个提示，以便它能跟踪每个节点的身份，从而重用和重新排序现有元素，你需要为每项提供一个唯一的 key 属性：

```
<div id="hello-vue">
  <ul>
    <li v-for="(v, k) in myObject" :key="k">
      {{ k }}-{{ v }}
    </li>
  </ul>
</div>
<script type="text/javascript">
    const HelloVueApp = {
      data() {
        return {
            myObject: {
                host: '主机',
                displayer: '显示器',
                keyboard: '键盘'
            }
        }
      }
    }
    Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```

v-for：选择列表案例

获取用户选择并赋值另一个变量再实时展示：

```
<div id="hello-vue">
    <select @change="selectComputer($event)">
        <option value="None">未选择</option>
        <option v-for="row in computer" :key="row.id" :value="row.id">
            {{row.name}}
        </option>
    </select>
    <p>当前选择主机ID：{{ selectComputerId }}</p>
</div>
<script type="text/javascript">
    const HelloVueApp = {
      data() {
        return {
            computer: [
                {id:1, name: '主机1'},
                {id:2, name: '主机2'},
                {id:3, name: '主机3'},
            ],
            selectComputerId: ""
        }
      },
      methods: {
          selectComputer(event) {
              console.log(event)   // 获取该事件的事件对象
              this.selectComputerId = event.target.value;  // 获取事件的值
              if (this.selectComputerId == "None") {
                  this.selectComputerId = "未选择！"
              }
          }
      }
    }
    Vue.createApp(HelloVueApp).mount('#hello-vue')
</script>
```



效果图

谢谢

阿良个人微信

DevOps技术栈公众号

阿良教育：www.aliangedu.cn