4-2 直播 Web Components

模块问题答疑

Web Components

- 简介
  - 背景
  - 影响
  - 组件化开发
  - Web Components
- 使用
  - Custom elements 自定义元素
    - 使用方式
  - Shadow DOM 影子 DOM
    - 使用方式
    - 样式相关
  - HTML templates HTML 模板
    - 模板使用
    - 封装独立组件
  - slot 插槽
  - 封装 Button 组件
- 结语

内容介绍：

- 模块问题答疑
- Web Components

直播代码：📎 4-2 直播代码汇总.rar

模块问题答疑

1. 希望扩展一下路由跳转的方法，除了 to 还有别的吗？
2. 讲一下那个编程式导航，不明白
   a. **回顾功能**：Vue Router 注册为 Vue 插件后给 Vue 实例添加的两个属性，其中 $route 存储了路由相关信息，$router 代表存储路由的操作方法。

b. **回顾操作方式**：
    i. <router-link> 与 <router-view> 结合，可通过 <router-link> 的 to 属性设置目标路由地址。
    ii. <router-view> 与 $router 的方法结合，可通过 $router.push() 设置目标路由地址，这种使用方式称为**编程式导航**。
        1. 概念：编程式导航指的是**通过编程写法进行导航操作**。编程式，就是书写代码而不是标签。
        2. 场景：例如，当导航操作是有条件的进行的，就需要使用编程式导航，例如登录成功的导航。总的来说，<router-link> 就是一个用来快捷设置导航操作的组件而已。
c. **其他方法**：
    i. 见文档：地址

3. MVVM 中， Model 是后端的数据吗？我认为 ViewModel（ Vue 实例）中的 data 只是Vue的状态数据而已，应该不是MVVM框架中的model。
    a. 回顾概念：
        i. M（Model）模型，V（View）视图，VM（ViewModel）绑定 M 与 V。
        ii. **通俗的讲**，MVVM 是由开发者提供 M 与 V，并由 VM 自动将两者结合的一种软件架构方式。
    b. 问题解答：
        i. 状态就是一种数据，状态可以由用户的本地操作决定，也可以由后端响应数据决定，本身并不冲突。
        ii. Vue 实例的 data 属性中可以存储一切与当前组件有关的数据，无论这个数据是什么，从哪来。
        iii. **可能**导致疑惑的点：
            1. Vue.js 文档中曾指出，Vue“没有完全遵循 MVVM 模型”，指的是 $refs 的用法会让 M 越过 VM 直接操作 V，这一点违反了 MVVM 模型的原则。
4. 路由 & 导航 两个名词的区别。
    a. 路由：路由是名词，在 Vue Router 部分简单理解为**地址**与**组件**的**对应关系**，这种关系称为路由。
    b. 导航：导航是动词，指的是我们从一个路由跳转到另一个路由的操作，这个跳转行为称为导航。
    c. 举例
        i. 动态路由：指的是一种路由关系的书写方式。
        ii. 编程式导航：指的是跳转路由的书写方式为编程写法。
        iii. 导航守卫：导航守卫是在路由跳转时触发的功能，钩子中可以接收 to 与 from 两个路由信息。
5. 说一下 Web Component，Vue官网中提到了
    a. Vue 文档中提及的位置：地址

Web Components

Web Components 是 W3C 组件指定的组件规范。

我们将分为以下几步进行讲解：

- 简介

- 基本使用

简介

抛开 Web Components，我们先来思考一个问题，是什么在推进技术的不断革新?
早期我们学习了 HTML、CSS、JavaScript，后来又学习了 jQuery，对于网页开发已经可以胜任了，为什么还会出现 Vue 等相关的框架并且推翻了 jQuery 长久的统治地位？不仅是 Vue，前端的海洋深不可测

背景

近十年可以说是 **Web 快速发展**的十年，随着网络的普及率大幅提高，生活中的一切仿佛都可以“上网”了。
以前手机充值需要去报亭或小卖部买充值卡；饿了想吃饭去饭馆，自己做饭要骑车去菜市场买菜；想购物了要坐车去商场，而且一个商铺通常只供应一种类型的商品，所以才会有苏宁电器（卖电器的）、Nike（卖服装的）、xx五金店（卖工具的），最惨的是公交卡没钱了还要去公交总站给公交卡充钱。
但是从现在来看，这一切都不是问题，我们只需要一部可以上网的手机就可以解决了，外卖网站可以帮我们吃到周边几公里内的所有美食，甚至有些商家还可以全城送货；电商网站则更加强大，可以帮我们买到一切我们需要（且合法）的商品，而范围是全世界。
现在的头部市场格局大体已定，各个公司就开始开发细分市场：买菜、买水果、充值、生活缴费、电影票、公园门票、看病挂号、报旅行团、甚至买车、买房都在网上可以解决。
小结：一切均“上网”，网上有一切。

影响

我们来思考一下，这种现实环境对于我们的网站开发会有什么样的影响呢?
丰富的业务需求导致 Web 应用的功能越来越多，功能复杂度越来越高，这就导致前端开发面临很多新的难题：
例如网页复杂度高，开发难度就高；传统写法导致功能与功能耦合，无法有效复用。
传统的 JavaScript 要求对代码进行模块化处理，可以有限的将功能与功能独立并实现复用，但随着功能日益增多，需要复用的就不仅仅是 JavaScript 了，还有配套的 HTML 与 CSS；由于 HTML、CSS 与 JavaScript 是没有直接关联（均是靠人为主观划分功能进行关联），所以当我们需要复用某个区域的功能时，例如一个列表，我们还是需要复制结构、封装样式文件、封装 JavaScript 文件，操作繁琐且单独的文件又增加的请求数。
由于现有的技术手段无法有效的解决问题，这就需要我们为前端**引入更多的新思想。**
小结：网站功能复杂度成倍增加，导致了一系列问题，为了解决问题需要引入新的手段。

组件化开发

为了实现网页功能层面的封装，前端引入了**组件**的概念，组件指的是某个网页功能区域的 HTML、CSS 与 JavaScript 的集合，将功能封装为组件可以有效的抽离功能，提高复用性与可维护性。
现代的前端开发的常用的第三方框架 Vue、React、Angular **均为组件化开发**方式，可见组件化开发是当前前端的主流开发方式，而传统的开发库如 jQuery 由于不能进行组件支持而不再适合开发大型 Web 应用。

小结：组件化开发时前端的发展方向。

Web Components

文档（MDN）：地址

Web Components 既 **Web 组件**。是 W3C 制定的 Web 组件规范，可以理解为浏览器的内置功能。

优点：

- Web 组件是浏览器的原生功能，无需引入任何第三方框架、模块即可使用。（减少项目体积）

缺点：

- 兼容性差（相比于 Vue）
- 还处于发展中，谨慎用于生产环境。

使用

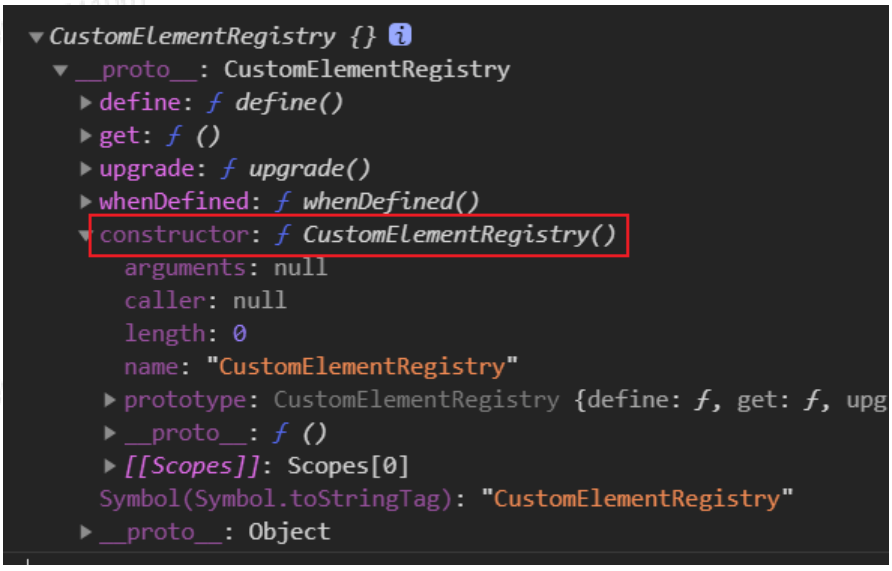
Custom elements 自定义元素

Web Components 的组件形式与 Vue 相似，都是**自定义元素**形式。

浏览器提供了 CustomElementRegistry 接口的实例来创建与操作自定义元素（组件），并在页面中按需使用。

在浏览器中，通过 **window.customElements** 属性来访问 CustomElementRegistry 实例。

- 可用于操作方法或访问创建的自定义元素。



使用方式

customElements.define() 方法用于创建自定义元素。

参数：

- name：自定义元素名，命名必须采用**短横线命名法**（kebab-case）。
- class：用于定义元素的**类**（ES 2015 语法）

- options：**可选**的配置对象，目前仅支持 extends 属性，用于指定创建的元素继承的内置元素的元素名。

使用示例：

1. Autonomous custom elements

指的是独立的、不继承自内置元素的自定义元素，独立元素的类必须继承自 HTMLElement。

```
// 1 首先，创建类用于定义元素
//   - 自定义元素必须继承自 HTMLElement，否则连元素的基本功能都没有了
class DemoElement extends HTMLElement {
  constructor () {
    // 继承则必须调用 super()
    super()
    // 此处不能直接操作，后续讲解操作方式
  }
}

// 创建自定义元素
window.customElements.define('demo-element', DemoElement)
```

在 HTML 中使用

```
<demo-element></demo-element>
```

2. Customized built-in elements

指的是继承子内置元素的自定义元素，在类继承与组件书写时有所不同。

```
// 设置用于定义自定义元素的类
class DemoElement extends HTMLUListElement {
  constructor () {
    super()
    // 可以直接进行 DOM 操作
    const image = document.createElement('img')
    image.src = 'https://s21.lgstatic.com/growth/activity/20210128/1611825307111.png'
    image.height = '200'
    this.append(image)
  }
}
```

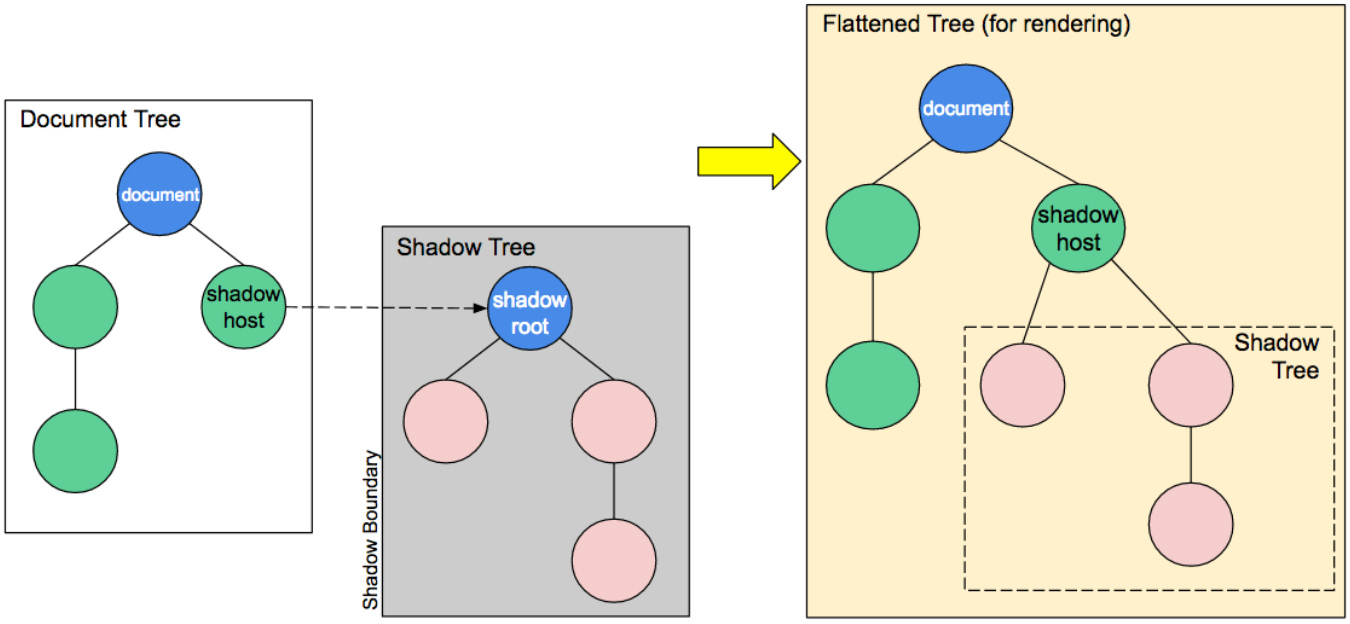
```
12
13 // 创建自定义元素，并指定继承的元素
14 window.customElements.define('demo-element', DemoElement, { exten
   ds: 'ul' })
```

在 HTML 中使用

```
1 <ul is="demo-element"></ul>
```

Shadow DOM 影子 DOM

Shadow DOM 指的是为组件所创建的独立的 DOM 树，它与文档中默认存在的 DOM 是独立的。



相关术语：

- Shadow host：一个常规 DOM 节点，Shadow DOM 会被附加到这个节点上。
- Shadow tree：Shadow DOM 内部的 DOM 树。
- Shadow boundary（边界）：Shadow DOM 结束的地方，也是常规 DOM 开始的地方。
- Shadow root：Shadow tree 的根节点。

总的来说，Shadow DOM 只是浏览器内部用于确保组件独立性的一种设置，可以确保组件内部不受外界影响，但在操作上**与常规 DOM 是相同的**。

使用方式

通过 Element.attachShadow() 来为元素添加 Shadow root。

参数：

- options：配置对象，可通过 mode 属性设置 Shadow DOM 是否可以被外界访问。
  - 可选值 'open'、'closed'

- 'open' 时，可通过 Element.shadowRoot 获取到 Shadow tree 的根节点，并进一步操作内部元素。
- 'closed' 时，Element.shadowRoot 为 null，无法获取到 Shadow DOM。

```
// 设置用于定义自定义元素的类
class DemoElement extends HTMLElement {
  constructor () {
    super()
    // 为元素创建 Shadow
    const shadow = this.attachShadow({ mode: 'open' })
    // 进行元素内容设置
    const image = document.createElement('img')
    image.src = 'https://s21.lgstatic.com/growth/activity/20210128/1611825307111.png'
    image.height = '200'
    // 添加时给 shadow 设置，而不是设置给元素。
    shadow.append(image)
  }
}

// 创建自定义元素
window.customElements.define('demo-element', DemoElement)
```

可更改 mode 属性，并尝试通过元素访问 shadowRoot 属性观察访问情况。

样式相关

除了可以设置结构，样式可以设置，且 Shadow DOM 让**组件内的样式与外部独立**。

```
// 设置用于定义自定义元素的类
class DemoElement extends HTMLElement {
  constructor () {
    // ...
    const text = document.createElement('p')
    text.textContent =  '组件内容'

    // 添加样式
    const style = document.createElement('style')
    style.textContent = `
      p {
```

```
        font-size: 20px;
        color: blue;
      }
    `
    // 添加时给 shadow 设置，而不是设置给元素。
    shadow.append(image, text, style)
  }
}
```

HTML 中进行以下设置

- 外部样式只对外部生效，内部样式只对内部生效

```
<style>
  p {
    text-decoration: underline;
  }
</style>
...
<body>
  ...
  <demo-element></demo-element>
  <p>组件外的 p 元素</p>
</body>
```

在 shadow DOM 中可以使用选择器 `:host` 用于虎丘 Shadow Host 元素（在 Shadow DOM 外无效）。

```

style.textContent = `
  :host {
    border: 1px solid blue;
  }
  ...
`
```

HTML templates HTML 模板

在前面的示例中我们会发现结构与样式的书写较为繁琐，且这种结构无法进行复用，这时就可以通过 HTML
templates 来进行处理。
HTML 模板本质是将组件的结构与样式书写在 <template> 标签中，由于 <template> 本身只是存储结构与样式的容器，所以在页面加载时 <template> 的内容并不会渲染。
我们可以给 <template> 设置 id 属性，从而获取到模板，再进一步通过 content 属性获取内部的 DocumentFragment 实例进行元素操作。

模板使用

设置 HTML 模板

```
<body>
  <template id="myTemplate">
    <style>
      p {
        font-size: 20px;
        color: blue;
      }
    </style>
    <img
      src="https://s21.lgstatic.com/growth/activity/20210128/1611825307111.png"
      height="200"
    >
    <h2>组件标题</h2>
    <p>组件内容</p>
    <ul>
      <li>列表项1</li>
      <li>列表项2</li>
      <li>列表项3</li>
    </ul>
  </template>
    ...
</body>
```

创建自定义元素，并引入模板

```
// 设置用于定义自定义元素的类
class DemoElement extends HTMLElement {
```

```
  constructor () {
    super()
    // 给元素创建 Shadow
    const shadow = this.attachShadow({ mode: 'open' })
    // 获取模板
    const template = document.getElementById('myTemplate')
    // 将模板元素的副本添加到 Shadow 中，使用副本是考虑组件复用的情况
    //   - 模板的 content 属性用于获取模板内的 DocumentFragment
    shadow.append(template.content.cloneNode(true))
  }
}

// 创建自定义元素
window.customElements.define('demo-element', DemoElement)
```

封装独立组件

按照上面的写法，一个组件就分为了两个部分，结构与样式保存再 HTML 中的 template 内，js 单独封装为文件。

学习 Vue CLI 后，我们发现单文件组件的书写方式的可维护性更高，所以为了增强组件的独立性，可以将 template 通过 js 创建，这样即可将组件功能完全独立封装起来了。

```
// 创建 template 标签
const template = document.createElement('template')
// 设置 template 内容（无需书写 template 标签部分）
template.innerHTML = `
  <style>
    p {
      font-size: 20px;
      color: blue;
    }
  </style>
  <img
    src="https://s21.lgstatic.com/growth/activity/20210128/1611825307111.png"
    height="200"
  >
  <h2>组件标题</h2>
  <p>组件内容</p>
```

```
  <ul>
    <li>列表项1</li>
    <li>列表项2</li>
    <li>列表项3</li>
  </ul>
`

// 创建自定义元素
class DemoElement extends HTMLElement {
  constructor () {
    super()
    // 创建 shadow
    const shadow = this.attachShadow({ mode: 'open' })
    // 直接通过 template 访问，无需根据 id 获取
    shadow.append(template.content.cloneNode(true))
  }
}
window.customElements.define('demo-element', DemoElement)
```

slot 插槽

与 Vue 相似，当我们希望对组件中的部分元素进行自定义设置时，就需要通过插槽来实现。

使用方式如下：仅为核心代码

1. 组件模板内通过 <slot> 使用插槽，name 属性用于选择指定插槽。
    a. 如果只有一个插槽则可以不写 name。

```
template.innerHTML = `
  <img
    src="https://s21.lgstatic.com/growth/activity/20210128/1611825307111.png"
    height="150"
  >
  <h2>组件标题</h2>
  <slot name="content"></slot>
`
```

2. 在 HTML 中使用组件时，可在自定义元素内设置元素，并通过 slot 属性将元素设置为指定插槽内容。
    a. 如果只有一个插槽，则自定义元素内部的内容会自动成为插槽内容。

```html
<demo-element>
  <div slot="content">
    <a href="#">内容1</a>
  </div>
</demo-element>
<demo-element>
  <div slot="content">
    <span>内容2</span>
  </div>
</demo-element>
```

封装 Button 组件

```
const template = document.createElement('template')
template.innerHTML = `
  <slot></slot>
`

class Button extends HTMLElement {
  constructor() {
    super()
    // 创建 Shadow
    const shadowRoot = this.attachShadow({ mode: 'open' })
    // 创建按钮
    const btn = document.createElement('button')
    // 将插槽内容设置给按钮
    btn.appendChild(template.content.cloneNode(true))
    // 定义按钮类型与对应颜色
    const colorOfTypes = {
      primary: 'blue',
      success: 'green',
      info: 'grey',
      warning: 'orange',
      danger: 'red'
    }
    // 获取按钮类型，默认为 primary
    const currentType = this.getAttribute('type') || 'primary'
    // 根据类型给按钮设置颜色
```

```
    btn.style.backgroundColor = colorOfTypes[currentType]

    // 创建并设置按钮基础样式
    const style = document.createElement('style')
    style.textContent = `
      button {
        border: 0 none;
        outline-style: none;
        padding: 10px 30px;
        border-radius: 20px;
        color: #fff;
        cursor: pointer;
      }
      button:hover {
        opacity: 0.9;
      }
      button:active {
        opacity: 0.8;
      }
    `
    shadowRoot.append(btn, style)
  }
}
customElements.define('lagou-button', Button);
```

在 HTML 中测试组件使用

```
<lagou-button type="primary" id="btn1">按钮1</lagou-button>
<lagou-button type="success">按钮2</lagou-button>
<lagou-button type="info">按钮3</lagou-button>
<lagou-button type="warning">按钮4</lagou-button>
<lagou-button type="danger">按钮5</lagou-button>

<script>
  // DOM 操作与常规的 DOM 元素相同
  document.getElementById('btn1').onclick = function () { alert(1
) }
</script>
```

完成。

结语

以上为 Web Component 的使用说明。相比 Web 组件，Vue 等前端框架功能虽然需要单独引入第三方文件，但功能也更加强大，除组件外，数据绑定与数据通信均为开发提供了极大的便利。
由于 Web 组件还在发展中，现阶段无论是兼容还是性能都存在一定问题，但可以看到标准化组织推进 Web 组件的决心，希望未来可以实现更加强大、便捷的功能。

Good Hunting!