

Web Components

Make Data Think

以AI激活运维数据智慧，助力客户数字化转型

Web Components

分享人：王璨

Make Data Think

以AI激活运维数据智慧，助力客户数字化转型

// 引言

通常一个应用会以一棵嵌套的组件树（DOM tree）的形式来组织；

web组件 = html + css + js

组件特点：高内聚低耦合

拆分组件原则：可维护性+复用性



组件是前端的发展方向，现在流行的 React 和 Vue 都是组件框架。

react组件：

函数组件与 class 组件 => UI = fn(data)

vue组件：

vue单文件组件

谷歌公司由于掌握了 Chrome 浏览器，一直在推动浏览器的原生组件，即 Web Components API。相比第三方框架，原生组件简单直接，符合直觉，不用加载任何外部模块，代码量小。

目录

CONTENTS

web components介绍

web components运用

第三方封装库

总结



定义

Web Components 是一套不同的技术，允许您创建可重用的定制元素（它们的功能封装在您的代码之外）并且在您的web应用中使用它们。

-- MDN

// 展示效果



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>web components</title>
  </head>
  <body>
    <v-page>
      <v-header>标题</v-header>
      <v-content>
        <v-aside>侧边栏</v-aside>
        <v-main>内容区域</v-main>
      </v-content>
    </v-page>
  </body>
</html>
```

// 技术依赖



Web Components主要由三项技术组成，分别为

1. **Custom elements** (自定义元素)
2. **Shadow DOM** (影子DOM)
3. **HTML templates** (HTML模板)

它们可以一起使用来创建功能强大的定制元素，并且可以在我们喜欢的任何地方重用，不必担心代码冲突。

Web Components

运用

customElements

自定义标签

```
// 注册  
customElements.define('my-button', MyButton, extends:  
  Button)
```

- name 所创建的元素名称，且需符合 DOMString 标准的字符串。注意，custom element 的名称不能是单个单词，且其中必须要有短横线
- class 用于定义元素行为的类
- extends 可选参数，一个包含 extends 属性的配置对象，指定了所创建的元素继承自哪个内置元素，可以继承任何内置元素。



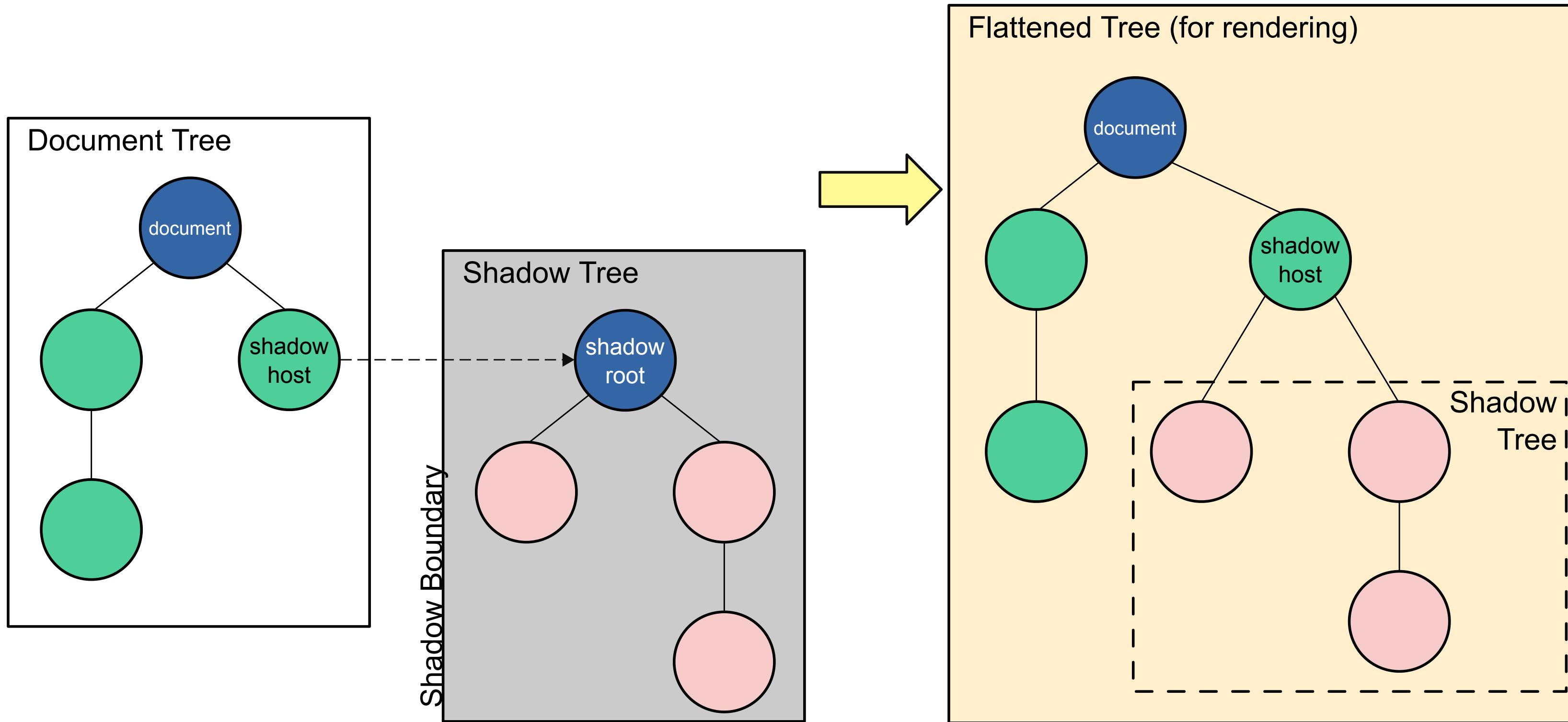
Shadow DOM

将一个隐藏的、独立的 DOM附加到一个元素上，并且允许将隐藏的 DOM 树附加到常规的 DOM 树中

```
class MyButton extends HTMLElement {  
  constructor() {  
    super();  
    // 第一步：创建组件根节点：shadow DOM节点  
    const shadow = this.attachShadow({ mode: 'open' })  
    // 第二步：向shadow DOM添加内容  
    shadow.innerHTML = `  
      <style>  
        <!-- 样式隔离 -->  
        button {  
          color: #fff;  
          background-color: #ccc;  
        }  
      </style>  
      <button>自定义按钮</button>  
    `;  
  }  
}
```



// shadow DOM



template

自定义模板，替代字符串模板

```
<template id="template">
  <style>
    button {
      color: #fff;
      background-color: #ccc;
    }
  </style>
  <button>自定义按钮</button>
</template>
```



// 组件插槽 - slot



```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>web components</title>
</head>
<body>
  <my-card>
    <p slot="username" class="name">张三</p>
  </my-card>
  <template id="my-card">
    <div class="container">
      <slot name="username"></slot>
    </div>
  </template>
  <script>
    class MyCard extends HTMLElement {
      ...
    }
    customElements.define('my-card', MyCard);
  </script>
</body>
</html>
```

// 生命周期 - Life Cycle



connectedCallback: 当 custom element首次被插入文档DOM时，被调用

disconnectedCallback: 当 custom element从文档DOM中删除时，被调用

adoptedCallback: 当 custom element被移动到新的文档时，被调用

attributeChangedCallback: 当 custom element增加、删除、修改自身属性时，被调用

// 组件通信 - 父到子



```
<!DOCTYPE html>
<html lang="en">
<body>
  <my-card name="张三"></my-card>
  <template id="my-card">
    ...
    <p class="name"></p>
  </template>
  <script>
    class MyCard extends HTMLElement {
      constructor() {
        ...
        const name = this.getAttribute('name');
        shadow.querySelector('.name').innerText = name;
      }
    }
    customElements.define('my-card', MyCard);
  </script>
</body>
</html>
```

// 组件通信 - 子到父



```
<script>
  class MyButton extends HTMLElement {
    constructor() {
      super();
      ...
      this.$button = shadow.querySelector('button');
      this.$button.addEventListener('click', () => {
        // 抛出事件
        this.dispatchEvent(new CustomEvent('custom', {
          detail: { message: '123' },
          composed: true // 表明当事件到达 shadow DOM 的根节点时，事件可以从 shadow DOM 传递到一般 DOM
        }));
      });
    }
    customElements.define('my-button', MyButton);
    const btn = document.querySelector('my-button');
    btn.addEventListener('custom', e => console.log(e.detail.message));
  }
</script>
```

// CustomEvent()

构造方法 CustomerEvent() 创建一个新的 CustomEvent 对象。

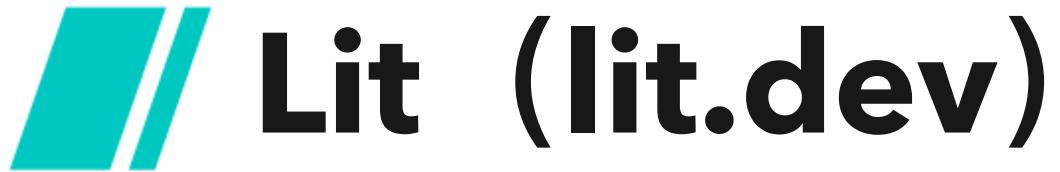
```
// add an appropriate event listener
obj.addEventListener("cat", function(e) {
    process(e.detail)
});

// create and dispatch the event
var event = new CustomEvent("cat", {
    detail: {
        hazcheeseburger: true
    }
});

obj.dispatchEvent(event);
```

第三方封装库

Lit / lit-element / vue-lit



谷歌开发，之前是 polymer，现在是 Lit

Lit 是一个简单的库，用于构建快速、轻量级的 web 组件（完全包含 lit-element, lit-html）

lit-element 是对原生自定义组件的一次封装，提供了一个可以创建 Web Component 的基类。可以类比于 jQuery 相对于 js

lit-html 是 lit-element 依赖的 HTML 模板引擎

vue-lit 是尤雨溪在去年9月，基于 vue3 @vue/reactivity 模块配合 lit-html 开发的一个可以直接在浏览器中渲染 vue 写法的 Web Component 的工具（70行代码）

lit-html 提供核心 render 能力

@vue/reactivity 提供 Vue 响应式系统的能力



```
import { html, render } from 'lit-html';
const Button = (text, props = { type: 'default', borderRadius: '2px' }, onClick) => {
  const clickHandler = {
    handleEvent(e) {
      alert('inner clicked!');
      if (onClick) {
        onClick();
      }
    },
    capture: true,
  };
  return html`
```

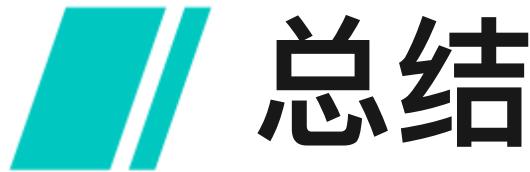


```
<!DOCTYPE html>
<html lang="en">
  <body>
    <my-component />
    <script type="module">
      import { defineComponent, reactive, html, onMounted } from 'https://unpkg.com/@vue/lit';
      defineComponent('my-component', () => {
        const state = reactive({ text: 'Hello World' });
        function onClick() { alert('clicked!') }
        onMounted(() => console.log('mounted'));
        return () => html`

<button @click=${onClick}>Click me</button>
          ${state.text}

`;
      })
    </script>
  </body>
</html>
```

总结



1. 创建一个类或函数来指定web组件的功能，如果使用类，请使用 ES6 的类语法；
2. 使用 CustomElementRegistry.define 方法注册您的自定义元素，并向其传递要定义的元素名称、指定元素功能的类、以及可选的其所继承自的元素；
3. 如果需要的话，使用 Element.attachShadow 方法将一个 shadow DOM 附加到自定义元素上。使用通常的 DOM 方法向 shadow DOM 中添加子元素、事件监听器等；
4. 如果需要的话，使用 template 和 slot 定义一个HTML 模板。再次使用常规DOM方法克隆模板并将其附加到您的 shadow DOM 中；
5. 在页面任何位置使用自定义元素，就像使用常规 HTML 元素那样。

优点：纯原生、无需编译、不依赖框架

缺点：需要操作DOM，缺少MV*框架中（VM）数据驱动视图的能力



Make Data Think

以AI激活运维数据智慧，助力客户数字化转型



www.eoitek.com



info@eoitek.com



4008 215 724