《Web 开发技术调研分析报告》
Microsoft Edge 从 React 到 Web Components

专业：计算机科学与技术
班级：2021211304
姓名：杨晨
学号：2021212171
提交日期：2025 年 1 月 7 日

1 新技术概述

随着 Web 应用程序的发展，JavaScript 在前端开发中扮演了至关重要的角色，尤其是通过框架如 React 的广泛应用。然而，近年来，JavaScript 生态系统的复杂性和对性能的影响引发了对更简洁、高效的开发方法的需求。Microsoft Edge 在其最新版本中首次推出了 “WebUI 2.0”，标志着从 React 转向 Web Components 的重大转变。这一转变旨在减少对 JavaScript 的依赖，提高浏览器的性能，特别是在低端设备上的表现。

Web Components 是一组基于 Web 平台的 API，允许开发者创建自定义、可重用且封装的 HTML 标签，用于在 Web 页面和 Web 应用程序中使用。它们的出现是为了提供更好的模块化和可维护性，解决传统 JavaScript 框架带来的复杂性和性能问题。Microsoft Edge 通过引入 WebUI 2.0，推动了 HTML 优先的开发方法，旨在简化开发流程，提高用户体验。

Web Components 包括四个核心技术：

- **自定义元素（Custom Elements）**：允许开发者定义新的 HTML 标签，并为其指定行为。
- **Shadow DOM**: 提供了封装的 DOM 和样式，防止外部样式和脚本影响组件内部。
- **HTML 模板（HTML Templates）**：定义了可重用的模板内容，便于在组件中进行插入和复用。
- **ES 模块（ES Modules）**：使组件的导入和管理更加高效和模块化。

这些技术共同构成了 Web Components，使其在现代 Web 开发中具备了强大的灵活性和可扩展性。Microsoft Edge 通过 WebUI 2.0 的引入，展示了 Web Components 在实际应用中的优势，尤其是在性能优化和开发效率提升方面。

2 新技术特点

2.1 技术特点

标记优先架构: WebUI 2.0 采用 HTML 优先的方法，减少了 JavaScript 在 UI 渲染中的作用，提高了页面加载和响应速度。这种架构强调使用纯 HTML 和 CSS 进行 UI 的构建，只有在必要时才引入 JavaScript，从而减少了不必要的代码量和复杂性。

Web Components: 利用 Web Components 创建可重用的 UI 组件，增强了模块化和封装性，简化了维护。通过自定义元素和 Shadow DOM，开发者可以将复杂的 UI 逻辑封装在独立的组件中，避免了全局命名空间的污染和样式冲突。

性能优化: 减少了 JavaScript 的使用，降低了浏览器的处理负担，提升了整体性能，尤其在低端设备上表现显著。通过减少客户端的脚本执行量，页面的加载时间和交互响应速度得到了明显改善。

兼容性: Web Components 作为 Web 标准的一部分，具有较高的跨浏览器兼容性，减少了对特定框架的依赖。主流浏览器如 Chrome、Firefox、Safari 和 Edge 都对 Web Components 提供了良好的支持，使得开发者能够编写一次代码，在多个浏览器上运行。

图 1: Microsoft 制作的简短视频演示速度差异

2.2 技术优势与劣势

优势:

- **性能提升**: 由于减少了 JavaScript 的使用，页面加载速度更快，响应更迅速。
- **简化开发**: 使用 Web Components 可以提高代码的可重用性和可维护性，减少开发复杂性。
- **更好的用户体验**: 尤其在低端设备上，用户界面更加流畅，减少了卡顿和延迟。
- **标准化和可移植性**: Web Components 基于 Web 标准，具有良好的跨平台和跨浏览器兼容性。

劣势:

- **迁移成本**: 现有使用 React 构建的应用程序需要进行大量重构，迁移成本较高。
- **学习曲线**: 开发团队需要掌握 Web Components 的相关知识，可能需要一定的学习时间。
- **功能限制**: 虽然 Web Components 具备高度的模块化，但在某些复杂交互和动态效果上，可能不如 React 等框架灵活。
- **工具和生态系统**: 相较于 React，Web Components 的开发工具和社区生态尚在发展中，资源可能相对有限。

3 新技术产品介绍与比较

3.1 Microsoft Edge WebUI 2.0

Microsoft Edge 在其版本 122 中引入了 WebUI 2.0，旨在通过 Web Components 替代 React 组件，提升浏览器性能。Edge 团队利用自家的 Fluent UI 框架，结合 Web Components，实现了更高效的 UI 渲染。WebUI 2.0 的核心目标是减少 JavaScript 的使用量，通过 HTML 和 CSS 实现更多的 UI 功能，从而提高整体性能和用户体验。

主要特性:

- **模块化组件**: 每个 UI 部分都被封装为独立的 Web Component，便于复用和维护。

- **性能优化**: 通过减少 JavaScript 的依赖，显著提升了页面加载和渲染速度。
- **兼容性提升**: 支持多种浏览器，确保不同用户的体验一致性。
- **易于维护**: 使用标准化的 Web Components，降低了代码复杂性，提高了可读性和可维护性。

开发工具与资源:

- **Fluent UI**: Microsoft 提供的统一设计系统，支持多种平台和框架，内置丰富的 UI 组件库。
- **Edge DevTools**: Edge 提供的开发者工具，支持调试 Web Components，性能分析和优化。
- **官方文档**: [Microsoft Edge WebUI 2.0 官方文档](https://docs.microsoft.com/en-us/microsoft-edge/webui-2-0) 提供了详细的技术指南和使用案例。

3.2 其他相关技术

React: 由 Facebook 开发的 JavaScript 库，用于构建用户界面，具有高度的灵活性和丰富的生态系统。React 通过虚拟 DOM 和组件化开发，极大地提高了开发效率和应用性能，广泛应用于各种 Web 应用中。

Web Components: 由 W3C 标准化，提供自定义元素、Shadow DOM 等功能，实现模块化和封装。Web Components 允许开发者创建独立、可复用的组件，增强了代码的可维护性和可扩展性。

SolidJS: 一个强调性能和响应性的 JavaScript 框架，提供与 Web Components 不同的开发体验。SolidJS 通过编译时优化和细粒度响应性，确保了高性能和高效的资源利用，适合构建复杂的应用程序。

3.3 比较分析

与 React 相比，Web Components 在性能和简洁性上具有明显优势，尤其是在减少 JavaScript 负担方面。然而，React 在生态系统和开发者支持方面更为成熟，提供了丰富的工具和库，适合构建复杂的应用程序。React 的虚拟 DOM 和丰富的生态系统使得它在构建大型应用时具有优势。

Web Components 则适用于需要高性能和模块化的场景，适合需要轻量级、可重用组件的项目。其基于标准的特性使得它在跨平台和跨浏览器兼容性方面表现出色，适合需要广泛兼容性的应用。然而，Web Components 的开发工具和社区生态仍在发展中，资源相对较少，可能限制了其在某些复杂应用中的应用。

特性	React	Web Components
性能	较高（虚拟 DOM）	更高（减少 JavaScript）
生态系统	丰富（大量库和工具）	较少（逐步发展中）
学习曲线	较低（广泛的学习资源）	较高（需要掌握标准 API）
兼容性	优良（通过 polyfill）	优良（原生支持）
可维护性	高（组件化）	高（标准化）

表 1: React 与 Web Components 比较表

4 新技术应用方法

以 Microsoft Edge 的 WebUI 2.0 为例，其应用方法如下：

4.1 安装与配置

1. **更新 Edge 浏览器**: 确保使用的是最新版本的 Microsoft Edge（版本 122 及以上）。可以通过[官方下载安装页面](https://www.microsoft.com/edge) 获取最新版本。
2. **启用 WebUI 2.0**: 在 Edge 的设置中启用 WebUI 2.0 功能。具体路径可能因版本不同而有所变化，通常可以通过以下步骤启用：
   - 打开 Edge 浏览器，点击右上角的“三点”菜单，选择“设置”。
   - 在设置页面中，导航到“系统和性能”。
   - 找到“WebUI 2.0”选项，切换为“启用”。
3. **引入 Fluent UI 框架**: 在项目中引入 Microsoft 的 Fluent UI 框架，以支持 Web Components 的使用。可以通过以下方式引入：
   - 使用 npm 安装 Fluent UI：

     ```
     npm install @fluentui/web-components
     ```

   - 在项目中导入 Fluent UI 组件：

     ```
     import { defineComponents } from '@fluentui/web-components';
     defineComponents();
     ```

4.2 使用方法

4.2.1 创建 Web Components

以下是创建一个简单的自定义元素的示例：

```
class MyComponent extends HTMLElement {
    constructor() {
        super();
        const shadow = this.attachShadow({ mode: 'open' });
        shadow.innerHTML = `
            <style>
                div {
                    padding: 10px;
                    background-color: #f0f0f0;
                    border: 1px solid #ccc;
                }
            </style>
            <div>这是一个自定义组件</div>
        `;
    }
}
```

```
customElements.define('my-component', MyComponent);
```

4.2.2 集成到项目中

将自定义元素集成到 HTML 文件中：

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <title>WebUI 2.0 示例</title>
    <script src="path/to/fluent-ui.js" defer></script>
    <script src="path/to/my-component.js" defer></script>
</head>
<body>
    <my-component></my-component>
</body>
</html>
```

4.2.3 优化与测试

1. **性能测试**: 使用 Edge 提供的开发者工具，评估 Web Components 的性能表现。可以通过以下步骤进行：
   - 打开 Edge 浏览器，按下 'F12'打开开发者工具。
   - 导航到“性能”面板，点击“录制”按钮，进行页面交互操作。
   - 停止录制后，分析 CPU 使用率、内存消耗和加载时间等指标，识别潜在的性能瓶颈。
2. **兼容性测试**: 确保 Web Components 在不同设备和浏览器上的兼容性。可以使用以下工具和方法：
   - **BrowserStack**: 在线跨浏览器测试工具，支持多种设备和浏览器组合。
   - **Lighthouse**: 开源自动化工具，提供性能、可访问性和最佳实践的评估报告。
3. **持续集成与部署**: 将 Web Components 集成到持续集成（CI）流程中，确保每次代码更新都经过自动化测试和性能评估。可以使用 Jenkins、Travis CI 或 GitHub Actions 等工具实现自动化。

4.3 高级应用方法

4.3.1 使用 Shadow DOM 进行样式封装

通过 Shadow DOM，可以为 Web Components 提供样式封装，避免外部样式的干扰。例如：

```
class StyledComponent extends HTMLElement {
    constructor() {
        super();
        const shadow = this.attachShadow({ mode: 'open' });
```

```
        shadow.innerHTML = `
            <style>
                p {
                    color: blue;
                    font-size: 16px;
                }
            </style>
            <p>这是一个带有封装样式的组件</p>
        `;
    }
}
customElements.define('styled-component', StyledComponent);
```

在上述示例中，‘<styled-component>‘的样式被封装在 Shadow DOM 中，外部样式不会影响组件内部的样式定义。

4.3.2 动态内容更新

利用 JavaScript 与 Web Components 的结合，实现动态内容的更新和交互：

```
class DynamicComponent extends HTMLElement {
    constructor() {
        super();
        this.attachShadow({ mode: 'open' });
        this.shadowRoot.innerHTML = `
            <button id="updateBtn">点击更新内容</button>
            <div id="content">初始内容</div>
        `;
        this.shadowRoot.getElementById('updateBtn').addEventListener('click', () =>
            {
            this.updateContent();
        });
    }

    updateContent() {
        this.shadowRoot.getElementById('content').textContent = '内容已更新';
    }
}
customElements.define('dynamic-component', DynamicComponent);
```

在上述示例中，点击按钮后，组件内部的内容会动态更新，展示了 Web Components 在处理用户交互方面的灵活性。

4.3.3 集成第三方库

在 Web Components 中集成第三方库，如图表库 Chart.js，实现高级功能：

```
import Chart from 'chart.js/auto';
```

```
class ChartComponent extends HTMLElement {
    constructor() {
        super();
        this.attachShadow({ mode: 'open' });
        this.shadowRoot.innerHTML = `
            <canvas id="myChart"></canvas>
        `;
    }

    connectedCallback() {
        const ctx = this.shadowRoot.getElementById('myChart').getContext('2d');
        new Chart(ctx, {
            type: 'bar',
            data: {
                labels: ['Red', 'Blue', 'Yellow'],
                datasets: [{
                    label: '#␣of␣Votes',
                    data: [12, 19, 3],
                    backgroundColor: ['red', 'blue', 'yellow']
                }]
            }
        });
    }
}
customElements.define('chart-component', ChartComponent);
```

通过集成 Chart.js，开发者可以在 Web Components 中轻松添加动态数据可视化功能，提升应用的互动性和数据展示效果。

4.3.4 与 React 的混合使用

虽然 Web Components 旨在替代某些 JavaScript 框架，但在实际开发中，开发者可以选择将其与 React 等框架混合使用，以发挥各自的优势。例如，可以在 React 应用中嵌入 Web Components，利用 React 的状态管理和组件化优势，同时享受 Web Components 的性能优化。

```
import React from 'react';
import './my-component.js'; // 引入自定义Web Component

function App() {
    return (
        <div>
            <h1>React与Web Components混合使用示例</h1>
            <my-component></my-component>
        </div>
    );
}
```

```
export default App;
```

这种混合使用的方法使得开发者可以逐步迁移现有的 React 组件到 Web Components，而无需一次性重构整个应用。

5 新技术应用领域解决方案

Web Components 作为一种模块化、可重用的前端技术，适用于多个应用领域，包括但不限于：

5.1 企业应用

在企业级 Web 应用中，Web Components 可以用于构建复杂的仪表盘和管理界面，通过模块化组件提高开发效率和维护性。例如，Microsoft Edge 团队在其浏览器 UI 中使用 WebUI 2.0，实现了更高效的界面渲染和更流畅的用户体验。企业应用通常需要处理大量数据和复杂的用户交互，Web Components 通过封装和复用，简化了开发流程，提高了代码的可维护性和可扩展性。

5.2 电子商务

电子商务网站可以利用 Web Components 创建可重用的商品展示、购物车和结算组件，提高页面加载速度和用户交互体验，特别是在移动设备上的表现尤为重要。通过 Web Components，开发者可以创建独立的商品卡片、推荐系统等模块，提升用户体验并简化开发流程。例如，一个电商平台可以创建一个自定义的“商品展示组件”，用于统一展示不同类别的商品，提高页面的一致性和加载效率。

5.3 内容管理系统

在内容管理系统（CMS）中，Web Components 可以用于构建可定制的内容编辑器和展示模块，提升编辑效率和内容呈现的灵活性。开发者可以创建独立的编辑器组件，如富文本编辑器、媒体上传组件等，集成到 CMS 中，提高内容管理的效率和用户体验。例如，WordPress 等流行的 CMS 平台可以通过 Web Components 集成自定义的内容模块，提升其功能和可扩展性。

5.4 跨平台应用

由于 Web Components 具有良好的跨浏览器兼容性，适用于开发需要在多个平台上运行的 Web 应用，减少了针对不同浏览器的适配工作。Web Components 的标准化特性使得开发者能够编写一次代码，在不同设备和浏览器上无缝运行。例如，一个跨平台的项目管理工具可以使用 Web Components 构建其核心 UI 组件，确保在桌面和移动设备上的一致性和性能。

5.5 教育与培训

在教育和培训领域，Web Components 可以用于创建互动式学习模块和可视化工具，提升学习效果和参与度。通过 Web Components，教育平台可以构建独立的教学模块，如测验组件、互动图表等，增强教学内容的互动性和趣味性。例如，一个在线学习平台可以使用 Web Components 创建自定义的测验题目组件，提供即时反馈和评分，提升学生的学习体验。

5.6 物联网（IoT）应用

在物联网应用中，Web Components 可以用于构建实时监控和控制界面，提供用户友好的操作体验。通过 Web Components，开发者可以创建独立的设备监控组件、控制面板等，简化开发流程，提高系统的响应速度和可靠性。例如，一个智能家居控制系统可以使用 Web Components 创建各类设备的控制界面，确保系统的高效和稳定运行。

6 新技术对社会与环境的影响

6.1 积极影响

提升用户体验: 更快的页面加载和更流畅的交互提升了整体用户体验，尤其是在资源受限的设备上。用户在使用 Web 应用时能够享受到更快速、更响应的界面，减少等待时间，提高满意度。

提高开发效率: 模块化和可重用的组件结构简化了开发流程，降低了维护成本。开发者可以通过复用组件，减少重复代码，提高开发效率，加快产品的迭代速度。

促进技术标准化: Web Components 作为 Web 标准的一部分，推动了前端技术的统一和标准化发展。标准化的组件开发方式使得不同团队和项目之间的协作更加顺畅，减少了技术栈的碎片化。

环境友好: 更高效的代码和优化的性能减少了服务器和客户端的资源消耗，降低了能源消耗，具有一定的环境友好性。通过优化前端性能，Web 应用的碳足迹得以减少，有助于可持续发展。

6.2 可能的负面影响

技术迁移成本: 大规模应用需要从 React 等框架迁移到 Web Components，可能带来高昂的时间和资源成本。企业和开发团队需要投入大量的时间和资金进行代码重构和培训，可能影响项目的进度和预算。

生态系统变化: 开发者需要适应新的开发模式，可能会导致短期内生产力下降。对于习惯于 React 生态系统的开发者来说，转向 Web Components 需要时间和精力，可能影响开发效率和项目进度。

浏览器依赖性: 尽管 Web Components 具备较高的兼容性，但不同浏览器对新标准的支持程度可能影响其广泛应用。部分老旧浏览器可能无法完全支持 Web Components，导致兼容性问题，影响用户体验。

安全性挑战: 封装性虽然提高了模块的安全性，但同时也可能带来新的安全挑战，如跨组件的通信和数据传输需要更加谨慎地处理，以防止潜在的安全漏洞。

7 总结与体会

Microsoft Edge 从 React 转向 Web Components 的举措，标志着前端开发领域的一次重要转变。这一转变不仅提升了浏览器的性能和用户体验，还为 Web 开发带来了更简洁、高效的开发模式。尽管迁移过程中面临一定的挑战，如技术迁移成本和开发者学习曲线，但 Web Components 作为 Web 标准的一部分，其模块化和可重用性为未来 Web 开发提供了坚实的基础。

随着 WebUI 2.0 的逐步推广和完善，预计更多的企业和开发团队将采用这种 HTML 优先的方法，推动 Web 开发向更高效、简洁的方向发展。未来，随着技术的不断进步和生态系统的成熟，Web Components 有望在更多应用场景中发挥重要作用，进一步提升 Web 应用的性能和用户体验。

此外，Web Components 的标准化和模块化特性也为跨团队协作和大型项目的开发提供了更好的支持，减少了不同项目之间的技术壁垒。随着社区对 Web Components 的认可和支持力度的加大，相关工具和库的生态系统将不断完善，进一步降低开发门槛，促进其在更广泛领域的应用。

总的来说，Web Components 代表了 Web 开发的一种未来方向，通过标准化、模块化和高性能的设计理念，为开发者提供了更多的灵活性和选择性，推动了 Web 技术的持续创新和发展。

致谢

感谢所有在 Web 开发领域做出贡献的开发者和社区成员，他们的努力和创新使得 Web 技术不断进步和完善。同时，感谢 Microsoft Edge 团队在 WebUI 2.0 项目中的辛勤工作和技术分享，为前端开发者提供了宝贵的经验和实践案例。

参考文献

[1] Ryan Carniato. “Web Components Are Not the Future”. In: *dev.to* (2024). URL: https://dev.to/ryansolid/web-components-are-not-the-future-48bh.

[2] Shubham Dhage. *Image on Unsplash: From React to HTML First: Microsoft Edge Debuts WebUI 2.0*. 2024. URL: https://unsplash.com/photos/c5909a00-shubham-dhage-4wk31qvi2pa.

[3] Cory LaViska. “Web Components Are Not the Future, They’ re the Present”. In: *A Beautiful Site* (2024). URL: https://www.abeautifulsite.net/posts/web-components-are-not-the-future-they-re-the-present/.

[4] Richard MacManus. “从 React 到 HTML 优先：Microsoft Edge 首次推出 “WebUI 2.0” ”. In: *The New Stack* (May 2024). URL: https://thenewstack.io/from-react-to-html-first-microsoft-edge-debuts-webui-2-0/.

[5] Kobe Mendes. *Image on Unsplash: Microsoft Edge 如何用 Web 组件替换 React*. 2024. URL: https://unsplash.com/photos/87a6c598-react-kick-1024x576.

[6] Andrew Ritz. *Andrew Ritz LinkedIn Profile*. 2024. URL: https://www.linkedin.com/in/andrewritz/.

[7] Alex Russell. *Response on Mastodon regarding WebUI 2.0*. 2024. URL: https://toot.cafe/@slightlyoff/112521248529973776.

[8] The New Stack. "2024 年 Web 开发趋势：回归简单". In: *The New Stack* (2024). URL: https://thenewstack.io/web-development-trends-in-2024-a-shift-back-to-simplicity/.

[9] The New Stack. *React to WebUI 2.0*. 2024. URL: https://www.youtube.com/embed/avJmgfGpoJA?feature=oembed.

[10] The New Stack. "可能影响前端开发人员的 2025 年虚拟主机趋势". In: *The New Stack* (2024). URL: https://thenewstack.io/2025-web-hosting-trends-that-could-affect-frontend-developers/.

[11] The New Stack. "帮助 Web 开发人员在 2025 年脱颖而出的 5 个技术趋势". In: *The New Stack* (2024). URL: https://thenewstack.io/5-technical-trends-to-help-web-developers-stand-out-in-2025/.

[12] The New Stack. "静态站点确实规模化：11ty 活动中的 11 对 Next.js". In: *The New Stack* (2024). URL: https://thenewstack.io/static-sites-do-scale-eleventy-vs-next-js-at-11ty-event/.

[13] WebComponents.org. *Introduction to Web Components*. 2024. URL: https://www.webcomponents.org/introduction.