

Инкапсуляция стилей и логики через Web Components: сравнение Shadow DOM с традиционными методологиями CSS

Мандронов Антон Андреевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье рассматриваются механизмы инкапсуляции стилей и поведения, предоставляемые технологией Shadow DOM в составе Web Components, в сравнении с традиционными методологиями организации CSS-кода (БЭМ, CSS Modules, CSS-in-JS). На основе тестового стенда, включающего пять вариантов реализации типовых UI-компонентов, представлены фрагменты кода, демонстрирующие ключевые различия в подходах к изоляции. Проведены замеры стилевых конфликтов, времени рендеринга и объёма ресурсов. Результаты показывают, что Shadow DOM обеспечивает наивысшую среди рассмотренных подходов степень изоляции (частичная, с возможностью полной), тогда как традиционные подходы не гарантируют защиты даже при строгом соблюдении соглашений. Производительность Web Components с теневым DOM при массовом создании оказалась вдвое ниже, чем у традиционных методологий, однако в типовых сценариях эта разница не критична. Предложена стратегия комбинированного выбора методологии в зависимости от масштаба проекта.

Ключевые слова: Web Components, Shadow DOM, инкапсуляция стилей, CSS, БЭМ, CSS Modules, CSS-in-JS, тестовый стенд, изоляция компонентов, веб-разработка.

Style and logic encapsulation via Web Components: comparing Shadow DOM with traditional CSS methodologies

Mandronov Anton Andreevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article examines the style and behavior encapsulation mechanisms provided by Shadow DOM technology within Web Components, in comparison with traditional CSS methodologies (BEM, CSS Modules, CSS-in-JS). Based on a test bench comprising five implementations of typical UI components, code snippets are presented that illustrate key differences in isolation approaches. Measurements of style conflicts, rendering time, and resource size are carried out. The results demonstrate that Shadow DOM provides the highest degree of isolation among the approaches considered (partial, upgradable to full), whereas traditional approaches

do not guarantee protection even with strict conventions. The performance of Web Components with Shadow DOM when instantiating in bulk was approximately twice lower than that of traditional methodologies, but in typical scenarios this difference is negligible. A combined strategy for choosing a methodology depending on project scale is proposed.

Keywords: Web Components, Shadow DOM, style encapsulation, CSS, BEM, CSS Modules, CSS-in-JS, test bench, component isolation, web development.

1 Введение

1.1 Актуальность

Рост сложности современных веб-приложений обостряет проблему управления стилями и логикой интерфейса. Глобальная природа каскадных таблиц стилей (CSS) приводит к непреднамеренным конфликтам селекторов, эффекту «призрачных стилей» и высокой когнитивной нагрузке при поддержке крупных кодовых баз. Традиционные методологии (БЭМ, CSS Modules) решают эти задачи на уровне соглашений, однако не предоставляют нативной изоляции на уровне браузера. С появлением стандарта Web Components и спецификации Shadow DOM стала возможной полная инкапсуляция DOM и стилей компонента. Сравнительный анализ этих подходов с привлечением конкретных реализаций и измерений необходим для обоснованного выбора архитектурного решения.

1.2 Обзор исследований

В работе И. В. Попкова и Л. В. Курзаевой проводится анализ методологии БЭМ как компонентного подхода к веб-разработке. Авторы приходят к выводу, что строгая система именования классов упорядочивает кодовую базу, но не способна полностью предотвратить конфликты глобальных стилей, что оставляет актуальной задачу поиска более надёжных механизмов изоляции [1]. В исследовании А. В. Максимченко рассматриваются особенности технологии Web Components API применительно к расширениям браузеров. Автор подчёркивает, что Shadow DOM предоставляет полноценную инкапсуляцию DOM и стилей, однако требует более тщательной проработки внутренней стилизации компонентов для достижения абсолютной независимости от внешнего контекста [2]. В публикации Д. Ю. Богомолова представлен сравнительный анализ способов создания встраиваемых веб-приложений, включая нативные компоненты и изоляцию через iframe [3]. В работе В. В. Чернова затрагивается фундаментальная проблема разработки веб-интерфейсов [4]. В исследовании И. М. Прохоровича и Д. В. Леунова выполнен обзор методологий создания веб-интерфейсов, таких как БЭМ, CSS Modules и Styled Components[5].

1.3 Цель исследования

Цель исследования — провести количественное и качественное сравнение инкапсуляции стилей и логики, обеспечиваемой Shadow DOM в Web Components, с традиционными методологиями CSS (БЭМ, CSS Modules,

CSS-in-JS) на основе единого тестового стенда с открытыми фрагментами кода, демонстрирующими все исследуемые реализации.

2 Материалы и методы

Для проведения сравнительного анализа разработан тестовый стенд, включающий три типовых UI-компонента: кнопка, карточка товара и модальное окно. Каждый компонент реализован в пяти вариантах, охватывающих основные подходы к управлению стилями:

- чистый HTML/CSS по методологии БЭМ (Vanilla JS);
- CSS Modules (сборка Vite, нативная поддержка);
- CSS-in-JS с библиотекой Emotion (генерация стилей во время выполнения);
- Web Component с открытым Shadow DOM (изоляция DOM и стилей);
- Web Component без Shadow DOM (Light DOM, для оценки эффекта инкапсуляции).

Все реализации дают идентичный визуальный результат; различия проявляются исключительно на уровне архитектуры и изоляции. Эксперименты проводились в браузере Google Chrome. Методика включала четыре группы измерений:

- Оценка утечки стилей под воздействием агрессивных глобальных правил.
- Анализ специфичности селекторов.
- Замер времени рендеринга 1000 экземпляров каждого варианта при эмуляции медленного процессора (6x throttling) с помощью Performance API.
- Сравнение объёма передаваемых ресурсов (HTML, CSS, JS) для страницы с 50 компонентами каждого типа.

Исходный код тестового стенда размещён в публичном репозитории, сборка выполнена инструментом Vite.

3 Результаты и обсуждения

Перед началом тестирования подготавливается единый HTML-документ, выступающий каркасом стенда. В нём определены контейнеры для каждого из пяти подходов, подключены внешние ресурсы (Emotion по CDN) и файлы стилей/скриптов, реализующие компоненты (рис. 1).

```

<!DOCTYPE html>
<html Lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Тестовый стенд: инкапсуляция стилей</title>
  <script src="https://unpkg.com/@emotion/css@11/dist/emotion-css.umd.min.js"></script>
  <style>
    body { font-family: sans-serif; margin: 20px; }
    [id$="-container"] {
      margin: 10px 0; padding: 10px; border: 1px dashed #gray;
    }
  </style>
  <link rel="stylesheet" href="./style-bem.css">
</head>
<body>
  <h2>Контейнеры для компонентов (карточка товара)</h2>
  <div id="bem-container"></div>
  <div id="cssmodules-container"></div>
  <div id="cssinjs-container"></div>
  <div id="shadow-container"></div>
  <div id="light-container"></div>

  <script src="./card-bem.js"></script>
  <script type="module" src="./cssmodules-card.js"></script>
  <script src="./emotion-card.js"></script>
  <script src="./product-card-shadow.js"></script>
  <script src="./product-card-light.js"></script>

</body>
</html>

```

Рис. 1. index.html

Компонент по методологии БЭМ реализован двумя файлами: style-bem.css (рис. 2) с глобальными классами и card-bem.js (рис. 3), который вставляет размеченный HTML в контейнер bem-container.

```

.bem-card {
  padding: 16px;
  border: 1px solid #ccc;
  background: white;
}
.bem-card__title {
  font-size: 18px;
  margin: 0;
}
.bem-card__price {
  color: green;
  font-weight: bold;
}

```

Рис. 2. style-bem.css

```
document.addEventListener('DOMContentLoaded', () => {
  const container = document.getElementById('bem-container');
  container.innerHTML = `
    <div class="bem-card">
      <h3 class="bem-card__title">Товар</h3>
      <span class="bem-card__price">1000 Р</span>
    </div>
  `;
});
```

Рис. 3. card-bem.js

Вариант с CSS Modules требует наличия сборщика (Vite). Стили описаны в файле Card.module.css (рис. 4), а скрипт cssmodules-card.js (рис. 5) импортирует их как объект и динамически присваивает сгенерированные имена классов элементу в cssmodules-container. В index.html данный скрипт подключён как модуль (type="module"), что позволяет использовать импорт.

```
.card {
  padding: 16px;
  border: 1px solid #ccc;
  background: white;
}
.title {
  font-size: 18px;
  margin: 0;
}
.price {
  color: green;
  font-weight: bold;
}
```

Рис. 4. Card.module.css

```
import styles from './Card.module.css';
const container = document.getElementById('cssmodules-container');
container.innerHTML = `
  <div class="${styles.card}">
    <h3 class="${styles.title}">Товар</h3>
    <span class="${styles.price}">1000 Р</span>
  </div>
`;
```

Рис. 5. cssmodules-card.js

CSS-in-JS решение использует библиотеку Emotion, загруженную через CDN. В файле emotion-card.js (рис. 6) создаются классы со случайными суффиксами, после чего HTML вставляется в cssinjs-container. Связь с index.html обеспечивается глобальным объектом emotion.

```
const { css } = emotion;

const cardClass = css`
  padding: 16px;
  border: 1px solid #ccc;
  background: white;
`;

const titleClass = css`
  font-size: 18px;
  margin: 0;
`;

const priceClass = css`
  color: green;
  font-weight: bold;
`;

document.getElementById('cssinjs-container').innerHTML = `
  <div class="${cardClass}">
    <h3 class="${titleClass}">Товар</h3>
    <span class="${priceClass}">1000 ₺</span>
  </div>
`;
```

Рис. 6. emotion-card.js

Web Component с теневым DOM определён в product-card-shadow.js (рис. 7). Он создаёт изолированное дерево, а в index.html достаточно вставить пользовательский тег <product-card> в shadow-container (выполняется в этом же файле при загрузке).

```
class ProductCard extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        .card { padding: 16px; border: 1px solid #ccc; background: white; }
        .title { font-size: 18px; margin: 0; }
        .price { color: green; font-weight: bold; }
      </style>
      <div class="card">
        <h3 class="title">Товар</h3>
        <span class="price">1000 ₺</span>
      </div>
    `;
  }
}

customElements.define('product-card', ProductCard);

document.addEventListener('DOMContentLoaded', () => {
  document.getElementById('shadow-container').innerHTML =
    '<product-card></product-card>';
});
```

Рис. 7. product-card-shadow.js

Аналог без Shadow DOM (рис. 8) вставляет стили и разметку непосредственно в Light DOM, имитируя отсутствие инкапсуляции. Компонент монтируется в light-container.

```
class ProductCardLight extends HTMLElement {
  constructor() {
    super();
  }
  connectedCallback() {
    if (!this.hasChildNodes()) {
      this.innerHTML = `
        <style>
          .card { padding: 16px; border: 1px solid #ccc; background: white; }
          .title { font-size: 18px; margin: 0; }
          .price { color: green; font-weight: bold; }
        </style>
        <div class="card">
          <h3 class="title">Товар</h3>
          <span class="price">1000 Р</span>
        </div>
      `;
    }
  }
}
customElements.define('product-card-light', ProductCardLight);
document.addEventListener('DOMContentLoaded', () => {
  document.getElementById('light-container').innerHTML =
    '<product-card-light></product-card-light>';
});
```

Рис. 8. product-card-light.js

После подключения всех файлов стенд визуально отображает пять идентичных карточек. Для проверки изоляции в index.html добавляется блок агрессивных глобальных стилей (рис. 9) и скрипт анализа вычисленных свойств (рис. 10). Соответствующий фрагмент встраивается прямо в основной документ.

```
<style id="aggressive-styles">
  div {
    color: red !important;
    background: yellow !important;
    border: 5px solid red !important;
  }
</style>
```

Рис. 9. Агрессивные стили

```
<script>
  window.addEventListener('DOMContentLoaded', () => {
    setTimeout(() => {
      function checkIsolation(el) {
        if (!el) return 'элемент не найден';
        const style = getComputedStyle(el);
        const colorOk = style.color !== 'rgb(255, 0, 0)';
        const bgOk = style.backgroundColor !== 'rgb(255, 255, 0)';
        if (colorOk && bgOk) return 'полная';
        if (colorOk || bgOk) return 'частичная';
        return 'пробита';
      }

      const bemEl = document.querySelector('#bem-container .bem-card');
      const cssmEl = document.querySelector(
        '#cssmodules-container div[class]',
      );
      const cssijEl = document.querySelector(
        '#cssinjs-container div[class]',
      );
      const wcShadowEl = document
        .querySelector('product-card')
        ?.shadowRoot?.querySelector('.card');
      const wcLightEl = document
        .querySelector('product-card-light')
        ?.querySelector('.card');

      console.log('БЭМ изоляция:', checkIsolation(bemEl));
      console.log('CSS Modules изоляция:', checkIsolation(cssmEl));
      console.log('CSS-in-JS изоляция:', checkIsolation(cssijEl));
      console.log('Shadow DOM изоляция:', checkIsolation(wcShadowEl));
      console.log('Light DOM изоляция:', checkIsolation(wcLightEl));
    }, 50);
  });
</script>
```

Рис. 10. Анализ вычисленных свойств

Для измерения времени рендеринга используется тот же HTML-документ. В конец `<body>` добавляется скрипт, динамически создающий 1000 экземпляров каждого варианта компонента и фиксирующий затраченное время с помощью Performance API (рис. 11). Все замеры проводятся при эмуляции медленного процессора (6x throttling в Chrome DevTools).

```

<script>
  window.addEventListener('DOMContentLoaded', () => {
    setTimeout(() => {
      function measureRender(factory, containerId, label) {
        const container = document.getElementById(containerId);
        if (!container) return;
        container.innerHTML = '';
        const start = performance.now();
        for (let i = 0; i < 1000; i++) {
          container.appendChild(factory());
        }
        const end = performance.now();
        console.log(`${label}: ${end - start}.toFixed(2)} ms`);
      }

      function bemFactory() {
        const div = document.createElement('div');
        div.className = 'bem-card';
        div.innerHTML = '<h3 class="bem-card__title">Товар</h3><span class="bem-card__price">1000 Р</span>';
        return div;
      }

      function cssModulesFactory() {
        const div = document.createElement('div');
        div.className = '_card_1a2b3c';
        div.innerHTML = '<h3 class="_title_1a2b3c">Товар</h3><span class="_price_1a2b3c">1000 Р</span>';
        return div;
      }

      function cssInJsFactory() {
        const { css } = emotion;
        const cardClass = css`padding: 16px; border: 1px solid #ccc; background: white;`;
        const titleClass = css`font-size: 18px; margin: 0;`;
        const priceClass = css`color: green; font-weight: bold;`;
        const div = document.createElement('div');
        div.className = cardClass;
        div.innerHTML = `<h3 class="${titleClass}">Товар</h3><span class="${priceClass}">1000 Р</span>`;
        return div;
      }

      function wcShadowFactory() {
        return document.createElement('product-card');
      }

      function wcLightFactory() {
        return document.createElement('product-card-light');
      }

      measureRender(bemFactory, 'bem-container', 'БЭМ');
      measureRender(cssModulesFactory, 'cssmodules-container', 'CSS Modules');
      measureRender(cssInJsFactory, 'cssinjs-container', 'CSS-in-JS');
      measureRender(wcShadowFactory, 'shadow-container', 'Shadow DOM');
      measureRender(wcLightFactory, 'light-container', 'Light DOM');
    }, 50);
  });
</script>

```

Рис. 11. Замер производительности (1000 экземпляров)

Сводные данные по изоляции и времени рендеринга, а также по объёму передаваемых ресурсов (измеренному отдельно) представлены в таблице (табл. 1).

Таблица 1 – Сравнительные характеристики методологий

Методология	Изоляция	Время рендера 1000 шт., мс
БЭМ	пробита	10.80
CSS Modules	пробита	8.20
CSS-in-JS (Emotion)	пробита	12.40
Web Component (Shadow DOM)	частичная	21.90
Web Component (Light DOM)	пробита	23.00

Анализ таблицы позволяет дать содержательную характеристику каждому подходу:

- БЭМ: время создания 1000 экземпляров — 10.80 мс, лучший показатель среди подходов без сборщика, но полная уязвимость к внешним стилям. Подход требует строгой дисциплины команды и не гарантирует изоляцию.

- CSS Modules: самая быстрая генерация (8.20 мс) благодаря тому, что на этапе сборки имена классов уже сгенерированы, а в рантайме просто присваивается строка. Однако изоляция также полностью пробита глобальными правилами.

- CSS-in-JS: умеренное время (12.40 мс), но изоляция пробита; уникальные имена классов не спасают от селекторов на теги. Дополнительный минус — наибольший размер бандла (29.4 КБ) из-за включённой runtime-библиотеки Emotion.

- Web Component с Shadow DOM: частичная изоляция (текст защищён, фон пробит), но время создания ощутимо выше — 21.90 мс, почти вдвое медленнее БЭМ. Это объясняется накладными расходами на создание теневого корня и инкапсулированных стилей для каждого экземпляра. При этом размер ресурсов минимален (17.5 КБ).

- Web Component без Shadow DOM (Light DOM): худшее время (23.00 мс) при полном отсутствии инкапсуляции. Выигрыш в объёме кода нивелируется отсутствием изоляции и низкой скоростью создания.

Таким образом, с точки зрения производительности при массовом создании однотипных элементов традиционные подходы (особенно CSS Modules) демонстрируют заметное преимущество перед Web Components — последние приблизительно вдвое медленнее. Однако в реалистичных сценариях, где на странице находятся десятки, а не тысячи компонентов, разница в ~10 мс остаётся не критичной. С позиции изоляции только Shadow DOM обеспечивает частичную защиту стилей, тогда как все прочие подходы не гарантируют изоляции даже при строгом соблюдении конвенций.

4 Выводы

Проведённое исследование позволяет сформулировать следующие выводы:

1. Shadow DOM в Web Components обеспечивает наивысшую среди рассмотренных подходов степень стилевой изоляции, хотя и неполную в рамках данного теста. Дополнительное явное задание всех CSS-свойств внутри теневого дерева способно довести её до абсолютной.

2. БЭМ, CSS Modules и CSS-in-JS не смогли противостоять глобальным стилям, нацеленным на HTML-теги, даже при строгой дисциплине именования. Следовательно, в условиях непредсказуемого окружения ни один из этих методов не гарантирует защиту стилей компонента.

3. По времени рендеринга 1000 экземпляров лидируют традиционные подходы: CSS Modules — 8.20 мс, БЭМ — 10.80 мс, CSS-in-JS — 12.40 мс. Web Components с Shadow DOM (21.90 мс) и без него (23.00 мс) оказались вдвое медленнее. В типовых приложениях с десятками компонентов данная разница не оказывает заметного влияния на пользовательский опыт.

4. CSS-in-JS продемонстрировал наибольший размер бандла (29.4 КБ), что критично для проектов с жёсткими требованиями к начальной загрузке. Нативные Web Components, напротив, не требуют дополнительных библиотек и имеют минимальный объём кода.

5. Нативная инкапсуляция Shadow DOM устраняет проблему каскадных конфликтов на уровне браузера, упрощает рефакторинг и снижает стоимость долгосрочной поддержки, однако разработчикам необходимо внимательно следить за полнотой стилового покрытия внутри теневого корня.

Предложена комбинированная стратегия: использование Shadow DOM для высокоуровневых составных компонентов, требующих надёжной изоляции, и традиционных методологий для простейших атомарных элементов, где важна максимальная скорость рендеринга.

Библиографический список

1. Попков И. В., Курзаева Л. В. БЭМ. Компонентный подход к веб-разработке //Международный студенческий научный вестник. 2018. №. 5. С. 165-165. URL: <https://s.eduherald.ru/pdf/2018/5/19170.pdf> (дата обращения: 24.04.2026)
2. Максимченко А. В. Особенности применения технологии Web Components API в расширениях браузеров: преимущества и ограничения //Актуальные исследования. 2025. №. 19 (254). С. 57-60. URL: <https://apni.ru/article/11978-osobennosti-primeneniya-tehnologii-web-components-api-v-rasshireniyah-brauzerov-preimushchestva-i-ogranicheniya> (дата обращения: 24.04.2026)
3. Богомоллов Д. Ю. Сравнительный анализ способов создания встраиваемых веб-приложений //Центральный научный вестник. 2019. Т. 4. №. 11. С. 13-14. URL: <https://elibrary.ru/item.asp?id=38224730> (дата обращения: 24.04.2026)

4. Чернов В. В. К проблеме разработки веб-интерфейсов //Фундаментальные исследования. 2012. №. 11-2. С. 463-465. URL: https://s.fundamental-research.ru/pdf/2012/2012_11_2.pdf#page=199 (дата обращения: 24.04.2026)
5. Прохорович И. М., Леунов Д. В. Анализ методологий для создания веб-интерфейсов //Международный студенческий научный вестник. 2018. №. 6. С. 85-85. URL: <https://s.eduherald.ru/pdf/2018/6/19283.pdf> (дата обращения: 24.04.2026)