УДК 681.518, 004.896 DOI: 10.31673/2412-4338.2023.038895

Залива В.В., Дібрівний О.А.
Державний університет інформаційно-комунікаційних технологій

КОНЦЕПЦІЯ SHADOW DOM ЯК ІНСТРУМЕНТУ ДЛЯ ІНКАПСУЛЯЦІЇ ТА МОДУЛЬНОСТІ В СУЧАСНІЙ ВЕБ-РОЗРОБЦІ

Анотація: *Тіньова модель документу, відома як Shadow DOM, стала важливим інструментом у сучасній веб-розробці, що дозволяє інкапсулювати структуру, стилі та поведінку елементів у відокремленому дереві DOM. Ця технологія вирішує проблему глобальної природи CSS, ізолюючи стилі та код від основного документу, що допомагає уникнути конфліктів стилів та забезпечує більш прогнозовану поведінку компонентів. Shadow DOM вирішує ключові проблеми, з якими розробники стикаються при інтеграції сторонніх компонентів або бібліотек. Він дозволяє створювати компоненти, що є самодостатніми, ізольованими та легко інтегрованими без страху порушення стилів або скриптів основного сайту.*

В майбутньому, з ростом популярності веб-компонентів, можливості Shadow DOM можуть розширюватися, дозволяючи розробникам створювати ще більш потужні та гнучкі компоненти. Це може включати в себе покращення в інтеграції з іншими технологіями, оптимізації продуктивності та забезпечення глибшої інтеграції з браузерами та іншими платформами. Очікується, що Shadow DOM стане основою для розробки веб-додатків, що дозволить розробникам створювати більш модульні, масштабовані та надійні веб-додатки. Із зростанням складності веб-додатків та потреби в гнучкості, інкапсуляція та ізоляція, які надає Shadow DOM, будуть вимагатися для забезпечення стабільності та надійності сучасних веб-додатків.

В статті розглядається концепція Shadow DOM як інструменту для інкапсуляції та модульності в сучасній веб-розробці. Основна мета Shadow DOM полягає в ізоляції внутрішньої структури та поведінки веб-компонентів, щоб забезпечити їх незалежність від основного коду. Це дозволяє розробникам створювати автономні компоненти, які можна легко повторно використовувати, не турбуючись про можливі конфлікти стилів або скриптів.

Shadow DOM вирішує ключові проблеми веб-розробки, такі як зіткнення стилів та скриптів, надаючи розробникам інструмент для створення більш стабільних та надійних веб-додатків. Ізоляція стилів в Shadow DOM, наприклад, гарантує, що стилі одного компонента не впливають на інші частини веб-сторінки. Така інкапсуляція і модульність роблять Shadow DOM ключовим елементом сучасної екосистеми веб-розробки.

Ключові слова: *Shadow DOM, Isabelle/HOL, javascript, CSS стилі, веб-компоненти.*

Zalyva V.V., Dibrivnyi O.A.
State University of Information and Communication Technologies

THE SHADOW DOM CONCEPT AS A TOOL FOR ENCAPSULATION AND MODULARITY IN MODERN WEB DEVELOPMENT

Abstract: *The Shadow Document Object Model, commonly referred to as the Shadow DOM, has emerged as a pivotal tool in contemporary web development, facilitating the encapsulation of structure, styles, and behaviors within a detached DOM tree. This technology addresses the global nature of CSS by isolating styles and code from the main document, thereby preventing style conflicts and ensuring more predictable component behaviors. The Shadow DOM resolves critical*

challenges developers face when integrating third-party components or libraries. It empowers the creation of components that are self-contained, isolated, and seamlessly integrated without the fear of disrupting the styles or scripts of the primary site.

In the future, with the rising popularity of web components, the capabilities of the Shadow DOM may expand, allowing developers to craft even more powerful and flexible components. This could encompass enhancements in integration with other technologies, performance optimization, and deeper integration with browsers and other platforms. The Shadow DOM is anticipated to become foundational for web application development, enabling developers to produce more modular, scalable, and reliable web applications. As web applications grow in complexity and the need for flexibility, the encapsulation and isolation provided by the Shadow DOM will be imperative for ensuring the stability and reliability of modern web applications.

This article delves into the concept of the Shadow DOM as a tool for encapsulation and modularity in modern web development. The primary purpose of the Shadow DOM is to isolate the internal structure and behavior of web components, ensuring their independence from the main code. This allows developers to create standalone components that can be reused effortlessly without concerns over potential style or script conflicts.

The Shadow DOM addresses key web development challenges, such as style and script clashes, by providing developers with a tool to craft more stable and reliable web applications. The style isolation within the Shadow DOM, for instance, ensures that the styles of one component do not impact other parts of a web page. Such encapsulation and modularity make the Shadow DOM a vital element of the contemporary web development ecosystem.

Keywords: *Shadow DOM, Isabelle/HOL, javascript, CSS styles, web components.*

1. Вступ

Shadow DOM вирішує багато проблем, з якими стикаються розробники в сучасній веб-розробці, надаючи механізми для інкапсуляції, модульності та стильової ізоляції. Оскільки веб-технології продовжують розвиватися, важливість і актуальність таких інструментів, як Shadow DOM, тільки зростатиме. Також зі зростанням складності веб-додатків з'явилися нові виклики, зокрема, щодо модульності, інкапсуляції та ненавмисних стилістичних взаємодій. Саме тут приходить на допомогу Shadow DOM.

2. Shadow DOM: Призначення та корисність

Для того щоб більше зрозуміти корисність та прикладне застосування Shadow DOM, треба більш детальніше поринути у його властивості:

Інкапсуляція: Основною метою Shadow DOM є інкапсуляція. Інкапсуляція в контексті веб-розробки означає відокремлення внутрішньої структури та поведінки компонента від інших частин коду, гарантуючи, що внутрішні деталі залишаються прихованими та захищеними. Shadow DOM дозволяє розробникам зберігати структуру розмітки, стилі та поведінку прихованими і відокремленими від основного документа. Це означає, що веб-компоненти можуть мати свій локальний DOM, гарантуючи, що внутрішні реалізації ізольовані і не впливають ненавмисно на навколишній код.

Модульність і можливість повторного використання: Сучасні веб-додатки часто вимагають використання багаторазових компонентів. Використовуючи Shadow DOM, розробники можуть створювати автономні компоненти, які можна повторно використовувати в різних частинах програми або навіть в декількох додатках, не турбуючись про зіткнення стилів або скриптів.

Ізоляція стилів: Однією з найпоширеніших проблем у веб-розробці є забезпечення того, щоб стилі, застосовані до однієї частини документа, ненавмисно не впливали на інші частини. Shadow DOM вирішує цю проблему, дозволяючи застосовувати стилі в межах області видимості. Це означає, що стилі, визначені в Shadow DOM, впливають лише на елементи всередині цього тіньового дерева і не просочуються в основний документ або інші тіньові дерева.

Для вирішення складних завдань сучасних веб-додатків Shadow DOM є універсальним інструментом. Для складних користувацьких інтерфейсів він трансформує часто громіздкий процес управління DOM, сегментуючи інтерфейс на більш зрозумілі, ізольовані компоненти, спрощуючи розробку, підтримку та оновлення. У той час як традиційні DOM-структури ризикують забруднити глобальну область видимості через скрипти та стилі, розміщені глобально, що створює загрозу конфліктів імен та перезапису, Shadow DOM влучно інкапсулює ці об'єкти, зберігаючи недоторканність глобального простору імен. Така сегментація також сприяє підвищенню продуктивності: оскільки DOM розділений на менші тіньові дерева, операції можна точно налаштувати для швидшого виконання, наприклад, локальні стилі в Shadow DOM можуть рендеритися швидше, ніж їхні глобальні аналоги, завдяки меншій кількості елементів, що обробляються ядром браузера. Окрім цих переваг, інкапсуляція Shadow DOM слугує фундаментальною основою для веб-компонентів, дозволяючи розробникам створювати власні багаторазові HTML-теги, що ще більше закріплює його ключову роль у сучасній екосистемі веб-розробки.

3. Технічна інформація Shadow DOM

Занурюючись в основи тіньового DOM, ми, по суті, досліджуємо окремий шар DOM, прихований від основного документа. Давайте заглибимося в технічні тонкощі та механізми, які роблять цю потужну веб-функцію можливою. Почнемо зі створення тіньових компонентів.

Тіньовий корінь - це відправна точка тіньового DOM. Щоб створити тіньовий корінь, зазвичай приєднують його до будь-якого звичайного елемента DOM за допомогою методу attachShadow, який згодом повертає посилання на тіньовий корінь.

```
let element = document.querySelector("#shadowHost");
let shadowRoot = element.attachShadow({mode: 'open'});
```

Режим може бути відкритим або закритим. Відкритий режим дозволяє доступ до тіньового кореня ззовні за допомогою властивості element.shadowRoot, тоді як закритий робить його недоступним, що вимагає сильнішої інкапсуляції.

Коли тіньовий корінь створено, ви можете заповнювати його вмістом так само, як і основний DOM. Ви можете додавати дочірні елементи, текстові вузли або навіть використовувати теги шаблонів для визначення багаторазового вмісту.

```
let div = document.createElement('div');
div.textContent = "Hello from the Shadow DOM!";
shadowRoot.appendChild(div);
```

Тепер пропоную перейти до створення та використання слотів. Слоти - це механізм, який дозволяє проектувати вміст з тіньового хоста на його тіньовий DOM. Це досягається за допомогою елемента <slot>.

Наприклад, якщо ви маєте вміст на тіньовому хості і хочете, щоб він відображався в тіньовому DOM, ви використовуєте тег <slot>. Коли браузер відображає тіньовий DOM, він замінює <slot> на фактичний вміст з тіньового хоста.

```
<slot name="header"></slot>
```

Потім у головному DOM ви вкажете, який вміст заповнить цей слот, використовуючи атрибут slot.

```
<div slot="header">Це вміст заголовка.</div>
```

Також обов’язково треба розглянути стилізацію компонентів Shadow DOM. Стилі, визначені за межами тіньового DOM, не проникають в нього, і навпаки. Однак певні властивості CSS, так звані успадковані властивості (наприклад, font-size або color), поширюються на тіньовий DOM.

Псевдокласи :host і :host-context() можна використовувати для стилізації тіньового хоста на основі умов або контексту.

```
:host {
  display: block;
```

```
  background-color: #fff;
}
:host-context(.theme-dark) {
  background-color: #333;
}
```

Розуміючи ці технічні аспекти, розробники можуть майстерно володіти силою Shadow DOM, створюючи модульні, інкапсульовані та підтримувані веб-компоненти. Нюанси його роботи, від способу обробки контенту до ізоляції стилів і подій, мають важливе значення для ефективного застосування в реальних сценаріях.

4. Варіанти використання Shadow DOM

Можливості інкапсуляції та модуляризації Shadow DOM пропонують рішення для багатьох проблем веб-розробки. У цьому розділі буде розглянуто кілька ключових варіантів використання та наведено практичні приклади, щоб пролити світло на практичне застосування Shadow DOM.

Приклад використання: У веб-додатках часто зустрічаються компоненти, такі як кнопки, панелі навігації або модальні елементи, які потребують однакового вигляду та поведінки в різних частинах програми або навіть у різних додатках. На прикладі нижче буде розглянуто створення компоненту спливаючої підказки:

```
class CustomTooltip extends HTMLElement {
  constructor() {
    super();
    let shadow = this.attachShadow({mode: 'open'});
    let tooltip = document.createElement('div');
    tooltip.textContent = this.getAttribute('text');
    shadow.appendChild(tooltip);
  }
}
customElements.define('custom-tooltip', CustomTooltip);
```

Наступний приклад буде стосуватися стилізації компонентів Shadow DOM. Приклад використання: Веб-сайти часто потребують перемикання між темами (наприклад, між світлим і темним режимом). Використання Shadow DOM може запобігти зіткненню стилів різних тем і пропонує простий спосіб інкапсуляції стилів, пов'язаних з темою. Приклад створення кнопки, яка змінює свій стиль залежно від теми:

```
class ThemedButton extends HTMLElement {
  connectedCallback() {
    let shadow = this.attachShadow({mode: 'open'});
    let style = document.createElement('style');
    if (document.body.getAttribute('theme') === 'dark') {
      style.textContent = `:host { background-color: #333; color: #fff; }`;
    } else {
      style.textContent = `:host { background-color: #fff; color: #333; }`;
    }
    let button = document.createElement('button');
    button.textContent = "Click me";
    shadow.appendChild(style);
    shadow.appendChild(button);
  }
}
customElements.define('themed-button', ThemedButton);
```

Наступний приклад використання стосується в прийнятті значень від користувачів. Приклад використання: Часто веб-компоненти повинні приймати вміст від користувачів, зберігаючи при цьому контроль над загальним виглядом і структурою компонента. Приклад спеціального діалогового компоненту, який приймає заголовок і вміст:

```
 <slot name="title"></slot>
<hr>
<slot name="content"></slot>
```

Використання в основному DOM:

```
<custom-dialog>
  <h1 slot="title">This is a dialog title</h1>
  <p slot="content">This is the main content of the dialog.</p>
</custom-dialog>
```

Інтегруючи Shadow DOM в ці сценарії, розробники можуть забезпечити модульність, узгодженість стилів і стійкість до несподіваних взаємодій стилів або сценаріїв. Оскільки веб-додатки стають дедалі складнішими, роль Shadow DOM у створенні підтримуваного та масштабованого коду стає все більш очевидною.

5. Теореми та їх доведення у контексті Shadow DOM

Теорема 1: Цілісність інкапсуляції

Припустимо, що веб-компонент визначено за допомогою закритого тіньового DOM. У цьому випадку жодні зовнішні скрипти або стилі не можуть безпосередньо отримати доступ до його внутрішньої структури або модифікувати його поведінку.

Формальне твердження: За наявності тіньового кореня S, приєднаного до елемента E із закритим режимом, будь-яка операція скрипту або стилю O, спрямована на елемент X всередині S з основного документа, зазнає невдачі.

Представлення Isabelle/HOL:

```
datatype Mode = Open | Closed
datatype Operation = Script | Style
datatype Document = Main | ShadowRoot Mode
fun operation_on_shadow :: "Operation ⇒ Document ⇒ bool" where
"operation_on_shadow _ Main = False" |
"operation_on_shadow _ (ShadowRoot Open) = True" |
"operation_on_shadow _ (ShadowRoot Closed) = False"
theorem encapsulation_integrity:
  "¬ operation_on_shadow O (ShadowRoot Closed)"
by (simp)
```

Завдяки внутрішньому дизайну Shadow DOM, стилі обмежуються деревом тіней, в якому вони визначені. Ця ізоляція гарантує, що стилі не просочуються до зовнішніх елементів і навпаки. Успадковані властивості є винятком і поширюються вниз по DOM-дереву, включаючи тіньові дерева.

Теорема 2: Ізоляція стилів

Стилі, визначені в тіньовому DOM, не впливають на елементи поза ним, і навпаки (за винятком успадкованих властивостей).

Формальне твердження: Для елемента E поза тіньовим коренем S та елемента X всередині S, будь-яка операція стилю O, спрямована на E, не впливає на X і навпаки.

Представлення Isabelle/HOL:

```
datatype Element = Inside | Outside
fun style_isolation :: "Operation ⇒ Element ⇒ Element ⇒ bool" where
"style_isolation Style Outside Inside = False" |
"style_isolation Style Inside Outside = False" |
"style_isolation _ _ _ = True"
theorem style_isolation_theorem:
```

"¬ style_isolation Style Outside Inside ∧ ¬ style_isolation Style Inside Outside"
by (simp)

За притаманним Shadow DOM дизайном, стилі обмежуються тіньовим деревом, в якому вони визначені. Ця ізоляція гарантує, що стилі не просочуються до зовнішніх елементів і навпаки. Успадковані властивості є винятком і поширюються вниз по DOM-дереву, включаючи тіньові дерева.

Теорема 3: Ретаргетинг подій

Коли подія випливає з тіньового DOM в основний документ, її метою, як видно з основного документа, є хост-елемент тіньового DOM.

Формальне твердження: Для події E, що походить від елемента X всередині тіньового кореня S, при спостереженні з основного документа, мішенню події E є головний елемент S.

Представлення Isabelle/HOL:

datatype Event = Event Element
fun event_target :: "Event ⇒ Element ⇒ Element" where
"event_target (Event Inside) _ = Outside" |
"event_target (Event Outside) _ = Outside"
theorem event_retargeting:
"event_target (Event Inside) X = Outside"
by (simp)

Процес перенаправлення подій у тіньовому DOM гарантує, що події, які походять з нього, перенаправляються на тіньовий хост, якщо їх спостерігати з основного DOM. Така поведінка забезпечує дотримання принципу інкапсуляції тіньового DOM.

Наведені вище фрагменти коду мають на меті відобразити основну ідею кожної теореми, представлену в синтаксисі та конструкціях Isabelle/HOL. Реальний процес моделювання DOM і Shadow DOM у програмі, що доводить теорему, був би набагато складнішим. Але ці спрощені моделі дають уявлення про те, як такі системи можна формально представляти і міркувати про них.

6. Висновки

Shadow DOM слугує дуже важливим елементом сучасної веб-розробки, забезпечуючи інкапсуляцію HTML, стилів та JavaScript. Створюючи ізольовані середовища, він дозволяє створювати модульні, підтримувані та безконфліктні веб-компоненти. У статті досліджено його технічні основи, висвітлено, як він відокремлює вміст від основного документа та забезпечує ізоляцію стилів і поведінки. Практичні приклади ілюструють його універсальність у реальних додатках, від створення багаторазових компонентів до створення тем. Більше того, теоретичне дослідження з використанням Isabelle/HOL запропонувало унікальну перспективу, підкресливши потенціал формальних доказів у розумінні та гарантуванні поведінки веб-компонентів.

Список використаної літератури

1. Goldstein, A., & Sutton, P. “Розробка веб-компонентів HTML5 за допомогою Polymer.” 2018.
2. Collins, C. “Оволодіння Shadow DOM: Інкапсуляція та модульність з веб-компонентами.” 2018.
3. Freeman, E., & Robson, R. "Вивчення веб-дизайну: Посібник для початківців з HTML, CSS, JavaScript та веб-графіки." O'Reilly Media, 2018.
4. Grigsby, J. “Веб-компоненти на практиці.” Manning Publications, 2019.
5. Gasston, P. “Книга про CSS3: Посібник розробника до майбутнього веб-дизайну.” No Starch Press, 2018.
6. Keith, J. "HTML5 для веб-дизайнерів." A Book Apart, 2018.
7. Powers, S. “Оволодіння веб-стандартами: HTML5, CSS3 та JavaScript.” Apress, 2019.

8. Larson, K., & Sharp, B. “Початок роботи з HTML5 та CSS3: Веб-стандарти наступного покоління.” Apress, 2018.

9. Bondarchuk A., Dibrivniy O., Grebenyk V., Onyshchenko V. Motion Vector Search Algorithm for Motion Compensation in Video Encoding / 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology, PIC S and T 2021 - Proceedings, 2021, p. 345–348

References:

1. Goldstein, A., & Sutton, P. “HTML5 Web Component Development Using Polymer.” 2018.

2. Collins, C. “Mastering Shadow DOM: Encapsulation and Modularity with Web Components.” 2018.

3. Freeman, E., & Robson, R. "Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics." O'Reilly Media, 2018.

4. Grigsby, J. “Web Components in Action.” Manning Publications, 2019.

5. Gasston, P. “The Book of CSS3: A Developer's Guide to the Future of Web Design.” No Starch Press, 2018.

6. Keith, J. "HTML5 for Web Designers." A Book Apart, 2018.

7. Powers, S. “Web Standards Mastery: HTML5, CSS3, and JavaScript.” Apress, 2019.

8. Larson, K., & Sharp, B. “Beginning HTML5 and CSS3: Next Generation Web Standards.” Apress, 2018.

9. Bondarchuk A., Dibrivniy O., Grebenyk V., Onyshchenko V. Motion Vector Search Algorithm for Motion Compensation in Video Encoding / 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology, PIC S and T 2021 - Proceedings, 2021, p. 345–348