Engineer

Вебкомпоненти

🖼 засоби браузера для створення власних HTML-елементів

Що це таке

Вебкомпоненти, англійською Web Components, — це загальна назва групи можливостей браузера, які дають розробникам змогу визначати власні елементи HTML. Сторінка потім використовує такий елемент, як будь-який інший тег. Стандарти називають складники власними іменами: custom elements та елементи `template` і `slot` описує Стандарт HTML, а тіньові дерева (shadow trees) — Стандарт DOM. Збірна назва походить із заголовка документа *Introduction to Web Components* — Working Group Note, яку W3C опублікував 24 липня 2014 року, а згодом припинив роботу над нею й позначив як документ, на який не варто посилатися.

Користувацький елемент (custom element) визначає клас JavaScript, що розширює `HTMLElement`, і його реєструють під певною назвою через `customElements.define()`. Визначення пояснює синтаксичному аналізатору HTML, як створювати елемент і як елемент має реагувати на зміни, і відтоді його можна створити з розмітки, через `document.createElement()` або через `new`. Назва починається з малої латинської літери ASCII, не містить великих літер ASCII й обов'язково містить дефіс. Дефіс відділяє власні назви від самої мови: Стандарт HTML обіцяє, що відтепер до HTML, SVG і MathML не додаватимуть елементів із дефісом у назві.

Є два різновиди. Автономний користувацький елемент (autonomous custom element), як-от `<flag-icon country="nl">` із самого стандарту, починає як звичайний елемент без вбудованого значення. Розширений вбудований елемент (customized built-in element) доповнює наявний елемент. Приклад стандарту — клас, що розширює `HTMLButtonElement`, реєструється з `{ extends: "button" }` і використовується як `<button is="plastic-button">`. Такий елемент зберігає все, що робить кнопку кнопкою, зокрема поведінку фокуса, надсилання форми й атрибут `disabled`.

Тіньове дерево — це окреме дерево вузлів, прикріплене до елемента, який називають його хостом; звичайне дерево хоста Стандарт DOM називає light tree. `attachShadow()` створює тіньовий корінь (shadow root) у режимі open або closed, і лише в режимі open властивість хоста `shadowRoot` його повертає. Елемент `slot` у тіньовому дереві хоста позначає місце, де з'являються власні дочірні елементи хоста. Він показує призначені йому вузли, а коли їх немає, то власний вміст, і іменований слот забирає ті дочірні елементи, чий атрибут `slot` містить його ім'я. Елемент `template` зберігає фрагмент HTML, який не відображається і який скрипт може клонувати. З атрибутом `shadowrootmode` елемент `template` представляє декларативний тіньовий корінь, тож тіньове дерево можна записати й просто в розмітці.

Застосуйте поняття до рішення

Застосуйте «Що таке вебкомпоненти?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Інтерфейси, що швидко завантажуються, працюють на телефоні й поводяться однаково в усіх браузерах ваших клієнтів.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Керувала full-stack-розробкою GIS-карт — full-stack-картографія — від бази даних до браузера

Керувала full-stack-розробкою GIS-карт Зв'язатися з нами →

Чому це важливо

Вигаданий тег колись мало що давав: Стандарт HTML зазначає, що такі елементи історично суперечили стандарту й були не надто функціональними. Користувацький елемент — повноправний учасник документа. Синтаксичний аналізатор його конструює, методи DOM його створюють, а його клас визначає, що відбувається, коли змінюються його атрибути або коли його вставляють на сторінку чи прибирають із неї.

Визначенню не обов'язково йти першим. Елемент, який синтаксичний аналізатор уже створив, стає користувацьким, щойно для його назви зареєструють визначення, і Стандарт HTML називає це оновленням (upgrade). Доти елемент вважається невизначеним, приблизно як `span`, тож текст і посилання в ньому показуються як є. Стандарт прямо каже, що оновлення дають змогу прогресивно покращувати вміст користувацьких елементів.

Користувацький елемент може також брати участь у формах і в дереві доступності. Клас, що задає `static formAssociated = true` і викликає `attachInternals()`, отримує об'єкт `ElementInternals`, а метод `setFormValue()` цього об'єкта дає елементу значення, яке форма надсилає так само, як і для будь-якого іншого елемента керування. Той самий об'єкт задає типову роль і типові стани ARIA, що діють, доки автор сторінки не перевизначить їх атрибутом `role` або відповідними атрибутами `aria-*`, а ще може тримати власні стани, до яких таблиці стилів звертаються псевдокласом `:state()`.

Для команди тіньове дерево цінне тим, що компонент стає самодостатнім: звичайні селектори сторінки не проникають у його внутрішнє тіньове дерево, а компонент сам вирішує, які частини відкрити для стилізації ззовні. Успадковані значення й custom properties усе ж можуть потрапляти всередину, а на вміст light tree, розміщений у слотах, і далі діють селектори сторінки. Там, де кілька бібліотек ділять одну сторінку, Стандарт HTML пропонує scoped registries, які створюють через `new CustomElementRegistry()`, щоб елементи цих бібліотек могли співіснувати й бібліотекам не доводилося узгоджувати між собою назви.

Як це виглядає на практиці

Картка товару показує, як складники працюють разом. Сторінка містить вміст як звичайну розмітку, з атрибутом `slot` на тому елементі, який має потрапити в іменований слот:

```
<product-card>
  <span slot="name">Дубовий табурет</span>
  <a href="/kramnytsia/dubovyi-taburet/">Детальніше</a>
</product-card>
<script type="module" src="product-card.mjs"></script>
```

Модуль визначає елемент. Шаблон будується один раз, і кожна картка отримує власну його копію у відкритому тіньовому корені:

```
const template = document.createElement("template");
template.innerHTML = `
  <h2 part="title"><slot name="name">Товар без назви</slot></h2>
  <slot></slot>
`;

class ProductCard extends HTMLElement {
```

```
  constructor() {
    super();
    const root = this.attachShadow({ mode: "open" });
    root.append(template.content.cloneNode(true));
  }
}

if (!customElements.get("product-card")) {
  customElements.define("product-card", ProductCard);
}
```

Сторінка стилізує заголовок через єдину частину, яку компонент відкриває назовні:

```
product-card::part(title) {
  font-size: 1.25rem;
}
```

Поки `product-card.mjs` не виконався, картка лишається невизначеним елементом, і читач бачить назву товару й посилання як звичайний вміст у тексті. Коли модуль виконується, `define()` реєструє клас, і картка, що вже є на сторінці, оновлюється. Конструктор прикріплює до неї тіньовий корінь і наповнює його клоном вмісту шаблону. Елемент `span` призначається слоту з назвою `name` і з'являється в заголовку. Посилання не має атрибута `slot`, тож потрапляє до безіменного слота за замовчуванням. Картка без назви натомість показує власний вміст слота, «Товар без назви». Перевірка через `customElements.get()` не дає зайвій копії модуля, що прийшла з іншим пакетом, змусити `define()` кинути помилку.

Спільні компоненти допомагають, лише коли ними справді користуються, і в портфоліо сайту є проєкт саме про це. Дизайн-система iOS 26 «Liquid Glass» починає з інтерфейсу, у якому кожен новий екран винаходить власні кнопки, бейджі й картки, щоразу трохи інакші, аж поки кожна зміна не вимагає правити п'ять саморобних версій того самого. Відповідь, яку описує сторінка, — канонічний набір компонентів, з яких має складатися кожен екран, серед них Card, Label, Button, GlassIconButton і DirectoryGrid, і правила лінтингу, що відхиляють саморобний бейдж чи чіп і вказують натомість на канонічний компонент. Сторінка не каже, чи ці компоненти є користувацькими елементами, але саму дисципліну можна перенести: одне визначення на

компонент, яке тримають інструменти, а не пам'ять рецензента. Реєстр користувацьких елементів забезпечує першу половину: він відхиляє нове визначення для назви, яку вже зайнято.

Де виникають проблеми

Назва з дефісом підміняє роль. Стандарт HTML каже прямо: «An autonomous custom element does not have any special meaning: it represents its children». Приклад стандарту `taco-button` показує, чого це коштує: щоб поводитися як кнопка, елементу потрібні `tabindex` , роль ARIA `button` , доступна назва, `aria-disabled` , обробка клавіатури, яка перетворює Enter і пробіл на клік, і стилі для вимкненого стану. Навіть тоді користувацький елемент не може задати типову поведінку фокуса, тож `tabindex="0"` може розходитися з тим, як кнопки поводяться на платформі. Там, де елемент спирається на наявний, стандарт каже, що розширений вбудований елемент легше розробляти, підтримувати й використовувати. Дайте автономному елементу типову роль і типові стани через `ElementInternals` , а не залишайте їх на кожну сторінку, що його використовує.

Конструктор робить усю роботу. Стандарт HTML ставить вимоги до конструкторів користувацьких елементів: першим іде виклик `super()` без аргументів; атрибути й дочірні елементи не можна перевіряти, бо створений, але не оновлений елемент їх не має; і конструктор не повинен додавати елементу атрибути чи дочірні елементи. Конструктор, що порушує ці правила, може дати елемент, який синтаксичний аналізатор або методи DOM не здатні створити. Використовуйте конструктор для початкового стану, типових значень, обробників подій і тіньового кореня, а завантаження й рендеринг перенесіть у `connectedCallback` .

Від `connectedCallback` чекають, що він спрацює один раз. Він може спрацювати й кілька разів. За замовчуванням переміщення елемента викликає `disconnectedCallback` , а потім `connectedCallback` , тож елемент скидає свій стан, наче його прибрали й вставили знову. Захищайте одноразове налаштування прапорцем і прибирайте в `disconnectedCallback` те, що запустив `connectedCallback` . Якщо елемент має зберігати стан, коли його переміщують через `moveBefore()` , визначте `connectedMoveCallback` : самої його наявності досить, щоб замінити типову пару, навіть із порожнім тілом.

Назву визначають двічі. `customElements.define()` кидає `NotSupportedError`, коли назву або клас уже зареєстровано, і `SyntaxError`, коли назва недійсна, наприклад через відсутній дефіс або велику літеру. Перевіряйте через `customElements.get()`, перш ніж визначати, а бібліотеці, яка має співіснувати з іншими, дайте власний scoped registry.

Режим closed вважають захистом. Режим closed ховає корінь від властивості хоста `shadowRoot` і приховує внутрішні вузли зі шляху події composed path для обробників, що перебувають поза цим закритим тіньовим деревом. CSS Shadow Parts прямо каже, що тіньові дерева навмисно не є межею безпеки, а лише зручністю для авторів сторінок. Тримайте секрети поза компонентами й обирайте режим closed заради його впливу на зовнішній код, а не як замок.

`attachShadow()` **викликають там, де він не може спрацювати.** Стандарт DOM дозволяє тіньовий корінь лише на елементах із дійсною назвою користувацького елемента та на короткому списку вбудованих елементів, серед яких `div`, `section`, `p` і `span`, але не `button`. На розширеній вбудованій кнопці виклик кидає `NotSupportedError`. Інакше він поводиться й на елементі, що вже має корінь: якщо корінь декларативний і в тому самому режимі, виклик очищає його й повертає, а якщо ні, то кидає помилку. Спершу перевіряйте, чи тіньовий корінь уже існує, щоб розмітку, яку сервер надіслав у декларативному корені, зберегти, а не стерти.

Від стилів сторінки чекають, що вони проникнуть усередину. CSS Shadow Parts називає цю межу бар'єром для стилів і зауважує, що вона ускладнює стилізацію компонента й тоді, коли сторінка справді цього хоче. Відкривайте назовні атрибутом `part` ті елементи, які сторінці дозволено стилізувати, і стилізуйте їх ззовні через `::part()`. Компонентові всередині іншого компонента потрібен `exportparts` на внутрішньому хості, щоб передати свої частини далі до сторінки. Для кольорів і відступів, які мають іти за сторінкою, використовуйте custom properties, задані на хості: вони успадковуються в тіньове дерево компонента.

Вміст з'являється лише після скрипту. Компонент, що пише весь свій текст із JavaScript, ховає його від читачів, доки не завантажиться визначення, а якщо скрипт зламається, читачі його так і не побачать. Тримайте вміст у light tree й призначайте його слотам, як це робить картка товару, щоб оновлення покращувало те, що вже є на сторінці. Там, де вміст мусить зачекати, власний

приклад Стандарту HTML ховає його, доки проміс від `customElements.whenDefined()` не буде виконано для кожного елемента, що відповідає селектору `:not(:defined)`.

Що почитати далі

- HTML Standard: Custom elements — правила назв, вимоги до конструкторів, оновлення, елементи, пов'язані з формами, і `ElementInternals`, з розібраними прикладами.
- DOM Standard: Shadow tree — тіньові корені та їхні хости, слоти й призначені їм вузли, а також кроки, що стоять за `attachShadow()`.
- HTML Standard: Scripting — елемент `template` з атрибутом `shadowrootmode` і елемент `slot`.
- CSS Shadow Parts — Working Draft W3C від 16 грудня 2025 року, що визначає `part`, `exportparts` і псевдоелемент `::part()`.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке вебкомпоненти?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/web-components/