Engineer

Composants web

🖼 la création d'éléments HTML réutilisables dans le navigateur

Ce que c’est

Les composants web, couramment appelés web components ou Web Components avec des majuscules, désignent un ensemble de fonctions du navigateur permettant aux auteurs de définir leurs propres éléments HTML. Une page utilise ensuite un tel élément comme toute autre balise. Les standards nomment séparément les différentes parties : les éléments personnalisés et les éléments `template` et `slot` dans le standard HTML, et les arbres fantômes dans le standard DOM. Leur nom collectif est le titre d’une note de groupe de travail du W3C datée du 24 juillet 2014, *Introduction to Web Components*, dont le W3C a depuis interrompu le développement en précisant qu’elle ne doit plus être citée en référence.

Un élément personnalisé est défini par une classe JavaScript qui étend `HTMLElement` et est enregistrée sous un nom avec `customElements.define()` . La définition indique à l’analyseur HTML comment construire l’élément et comment il doit réagir aux changements. Dès lors, l’élément peut être créé à partir de balisage, avec `document.createElement()` ou avec `new` . Son nom commence par une lettre ASCII minuscule, ne contient aucune lettre ASCII majuscule et doit contenir un trait d’union. Ce trait sépare les noms personnalisés du langage lui-même : le standard HTML précise que HTML, SVG et MathML n’acquerront aucun élément dont le nom contient un trait d’union.

Il en existe deux types. Un élément personnalisé autonome, tel que `<flag-icon country="nl">` dans le standard, part d’un élément simple sans signification intégrée. Un élément intégré personnalisé étend un élément existant : une classe qui étend `HTMLButtonElement` , enregistrée avec `{ extends: "button" }` et utilisée sous la forme `<button is="plastic-button">` , conserve tout ce qui fait d’un bouton un bouton, notamment le comportement du focus, l’envoi de formulaire et l’attribut `disabled` .

Un arbre fantôme est un arbre de nœuds distinct attaché à un élément appelé son hôte ; le standard DOM nomme light tree l'arbre ordinaire de cet hôte. `attachShadow()` crée une racine fantôme en mode ouvert ou fermé, et la propriété `shadowRoot` de l'hôte ne la renvoie qu'en mode ouvert. Un élément `slot` dans l'arbre fantôme indique où apparaissent les propres enfants de l'hôte. Il affiche les nœuds qui lui sont affectés, ou son propre contenu s'il n'y en a aucun. Un emplacement nommé reçoit les enfants dont l'attribut `slot` porte son nom. L'élément `template` contient un fragment de HTML qui n'est pas affiché et qu'un script peut cloner. Avec un attribut `shadowrootmode`, un `template` représente une racine fantôme déclarative ; un arbre fantôme peut donc aussi être écrit directement dans le balisage.

Utilisez le concept pour prendre une décision

Utilisez « Que sont les composants web ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des interfaces rapides, utilisables sur téléphone et cohérentes dans tous les navigateurs de vos clients.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Développement cartographique GIS full-stack — cartographie full-stack, de la base au navigateur

Développement cartographique GIS full-stack Contactez-nous →

Pourquoi c'est important

Une balise inventée était autrefois peu utile : le standard HTML décrit rétrospectivement de tels éléments comme invalides et peu utiles en pratique. Un élément personnalisé est un membre à part entière du document. L'analyseur le construit, les méthodes du DOM le créent et sa classe décide de ce qui se passe lorsque ses attributs changent ou lorsqu'il entre dans une page ou la quitte.

La définition n'a pas besoin d'arriver la première. Un élément déjà créé par l'analyseur devient un élément personnalisé lorsqu'une définition de son nom est enregistrée ; le standard HTML appelle cela une mise à niveau, upgrading. Jusque-là, il est traité comme un élément non défini, à la manière d'un `span` : le texte et les liens qu'il contient sont affichés tels quels. Le standard précise explicitement que les mises à niveau permettent d'améliorer progressivement le contenu d'un élément personnalisé.

Un élément personnalisé peut aussi participer aux formulaires et à l'arbre d'accessibilité. Une classe qui définit `static formAssociated = true` et appelle `attachInternals()` reçoit un objet `ElementInternals`. La méthode `setFormValue()` de cet objet donne à l'élément une valeur que son formulaire envoie comme celle de tout autre contrôle. Le même objet définit un rôle ARIA et des états ARIA prédéfinis, qui s'appliquent sauf si l'auteur de la page les remplace par l'attribut `role` ou les attributs `aria-*` correspondants. Il peut aussi conserver des états personnalisés que les feuilles de style ciblent avec la pseudo-classe `:state()`.

Pour une équipe, l'arbre fantôme rend le composant autonome : les sélecteurs ordinaires de la page n'entrent pas dans son arbre fantôme interne, et le composant décide des parties à exposer pour une mise en forme extérieure. Les valeurs héritées et les propriétés personnalisées peuvent encore y entrer, et le contenu du light tree affecté aux emplacements reste soumis aux sélecteurs de la page. Lorsque plusieurs bibliothèques partagent une page, le standard HTML propose des registres à portée limitée, créés avec `new CustomElementRegistry()`, pour que leurs éléments coexistent sans que les bibliothèques aient à coordonner leurs noms.

À quoi cela ressemble en pratique

Une fiche produit montre comment les différentes parties fonctionnent ensemble. La page contient le contenu sous forme de balisage ordinaire, avec un attribut `slot` sur l'élément destiné à un emplacement nommé :

```
<product-card>
  <span slot="name">Oak stool</span>
  <a href="/shop/oak-stool/">See the details</a>
</product-card>
<script type="module" src="product-card.mjs"></script>
```

Le module définit l'élément. Le modèle est construit une seule fois, et chaque fiche en reçoit sa propre copie dans une racine fantôme ouverte :

```
const template = document.createElement("template");
template.innerHTML = `
  <h2 part="title"><slot name="name">Unnamed product</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);
}
```

La page met en forme le titre par la seule partie que le composant expose :

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

Avant l'exécution de `product-card.mjs`, la fiche est un élément non défini et le lecteur voit le nom du produit et le lien comme du simple contenu en ligne. Lorsque le module s'exécute, `define()` enregistre la classe et la fiche déjà présente dans la page est mise à niveau. Son constructeur lui attache une racine fantôme et la remplit d'un clone du contenu du modèle. Le `span` est affecté à l'emplacement nommé `name` et apparaît dans le titre. Le lien ne possède pas d'attribut `slot` ; il va donc dans l'emplacement standard non nommé. Une fiche sans nom affiche à sa place le contenu propre de l'emplacement, « Unnamed product ». La vérification avec `customElements.get()` empêche qu'une seconde copie du module, arrivée dans un autre bundle, provoque une exception lors de l'appel à `define()`.

Des composants communs n'aident que s'ils sont utilisés, et le portfolio de ce site comporte une étude sur ce sujet. Le système de design iOS 26 « Liquid Glass » part d'une interface dans laquelle chaque nouvel écran réinvente ses boutons, badges et cartes, légèrement différents les uns des autres, jusqu'à ce que chaque modification oblige à toucher cinq versions sur mesure. La réponse décrite est un ensemble canonique de composants à partir duquel chaque écran se compose, notamment Card, Label, Button, GlassIconButton et DirectoryGrid, avec des règles de lint qui rejettent un badge ou une pastille fabriqués à la main et orientent vers le composant canonique. La page ne précise pas si ces composants sont des éléments personnalisés, mais la discipline se transpose : une définition

par composant, maintenue par l'outillage plutôt que par la mémoire de la personne chargée de la revue. Le registre d'éléments personnalisés impose l'unicité de la définition en refusant une seconde définition pour un même nom.

Où les problèmes surviennent

Un nom avec trait d'union tient lieu de rôle. « Un élément personnalisé autonome n'a aucune signification particulière : il représente ses enfants », précise le standard HTML. Son exemple `taco-button` montre le coût : pour se comporter comme un bouton, l'élément a besoin d'un `tabindex`, du rôle ARIA `button`, d'un nom accessible, d'`aria-disabled`, de gestionnaires de clavier transformant Entrée et Espace en clics et de styles pour l'état désactivé. Même alors, un élément personnalisé ne peut pas déclarer un comportement de focus prédéfini ; `tabindex="0"` peut donc ne pas correspondre au comportement des boutons sur la plateforme. Lorsqu'un élément s'appuie sur un élément existant, le standard indique qu'étendre un élément intégré réduit le travail de son auteur comme de ses utilisateurs. Pour un élément autonome, définissez son rôle et ses états prédéfinis avec `ElementInternals` au lieu d'en laisser la responsabilité à chaque page qui l'utilise.

Le constructeur effectue le travail. Le standard HTML impose des règles aux constructeurs d'éléments personnalisés : `super()` sans argument doit venir en premier ; le constructeur ne doit pas examiner les attributs ou les enfants de l'élément, car un élément créé plutôt que mis à niveau n'en possède aucun ; l'élément ne doit pas acquérir d'attributs ni d'enfants. Un constructeur qui enfreint ces règles peut produire un élément que l'analyseur ou les méthodes du DOM ne peuvent pas créer. Réservez le constructeur à l'état initial, aux valeurs prédéfinies, aux écouteurs d'événements et à la racine fantôme, et déplacez la récupération de données et le rendu dans `connectedCallback`.

On suppose que `connectedCallback` ne s'exécute qu'une fois. Il peut s'exécuter plusieurs fois. Le déplacement d'un élément déclenche normalement `disconnectedCallback`, puis `connectedCallback` : l'élément réinitialise donc son état comme s'il avait été retiré puis réinséré. Protégez avec un indicateur l'initialisation qui ne doit avoir lieu qu'une fois et annulez dans `disconnectedCallback` ce que `connectedCallback` a démarré. Si un élément doit conserver son état lorsqu'il est déplacé avec `moveBefore()`, définissez `connectedMoveCallback` : sa seule présence remplace la paire habituelle, même avec un corps vide.

Un nom est défini deux fois. `customElements.define()` lève une `NotSupportedError` lorsque le nom ou la classe est déjà enregistré, et une `SyntaxError` lorsque le nom est invalide, par exemple parce qu'il manque un trait d'union ou contient une majuscule. Vérifiez avec `customElements.get()` avant de définir et donnez son propre registre à portée limitée à une bibliothèque qui doit coexister avec d'autres.

Le mode fermé est pris pour une protection. Le mode fermé masque la racine à la propriété `shadowRoot` de l'hôte et masque les nœuds internes du chemin composé d'un événement tel qu'il est vu par les écouteurs situés en dehors de cet arbre fantôme fermé. CSS Shadow Parts précise que les arbres fantômes ne sont délibérément pas une frontière de sécurité, mais une commodité pour les auteurs de pages. Gardez les secrets hors des composants et choisissez le mode fermé pour son effet sur le code extérieur, pas comme un verrou.

`attachShadow()` **est appelé là où il ne peut pas fonctionner.** Le standard DOM n'autorise une racine fantôme que sur les éléments portant un nom d'élément personnalisé valide et sur une courte liste d'éléments intégrés, dont `div`, `section`, `p` et `span`, mais pas `button`. Sur un bouton intégré personnalisé, l'appel lève une `NotSupportedError`. Il se comporte aussi différemment sur un élément possédant déjà une racine : si celle-ci est déclarative et du même mode, l'appel la vide entièrement et la renvoie vide ; sinon, il lève une exception. Vérifiez d'abord l'existence d'une racine fantôme pour conserver le balisage envoyé par le serveur dans une racine déclarative, au lieu de l'effacer.

On s'attend à ce que les styles de la page atteignent l'intérieur. CSS Shadow Parts qualifie cette frontière de barrière de mise en forme et note qu'elle rend aussi un composant plus difficile à personnaliser lorsque la page le souhaite. Exposez les éléments qu'une page peut mettre en forme avec `part` et ciblez-les de l'extérieur avec `::part()`. Un composant imbriqué dans un autre a besoin d'`exportparts` sur l'hôte intérieur pour transmettre ses parties à la page. Pour les couleurs et espacements qui doivent suivre la page, lisez les propriétés personnalisées définies sur l'hôte, qui atteignent l'arbre fantôme par héritage.

Le contenu n'existe qu'après le script. Un composant qui écrit tout son texte avec JavaScript ne le montre pas aux lecteurs avant le chargement de sa définition, et ne le montre jamais si le script échoue. Gardez le contenu dans le light tree et affectez-le à des emplacements, comme le fait la fiche produit, pour que la mise à niveau améliore ce qui existe déjà. Lorsque le contenu doit

attendre, l'exemple du standard HTML le masque jusqu'à la résolution de `customElements.whenDefined()` pour chaque élément correspondant à `:not(:defined)` .

Pour aller plus loin

- Standard HTML : éléments personnalisés — règles de nommage, exigences des constructeurs, mises à niveau, éléments associés aux formulaires et `ElementInternals` , avec des exemples détaillés.
- Standard DOM : arbre fantôme — racines fantômes et leurs hôtes, emplacements et nœuds qui leur sont affectés, ainsi que les étapes d' `attachShadow()` .
- Standard HTML : scripts — l'élément `template` avec son attribut `shadowrootmode` et l'élément `slot` .
- CSS Shadow Parts — le projet de travail du W3C du 16 décembre 2025 qui définit `part` , `exportparts` et le pseudo-élément `::part()` .

Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Que sont les composants web ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/web-components/