Engineer

DOM

arbre d'objets d'une page, tenu à jour par le navigateur

Ce que c’est

Le DOM, Document Object Model, est l’arbre d’objets que le navigateur construit à partir d’une page et garde en mémoire tant qu’elle reste ouverte. L’analyseur HTML lit le balisage et produit un objet document. Chaque élément, chaque fragment de texte et chaque commentaire devient un nœud, qui connaît son parent, ses enfants et ses voisins. Les scripts lisent et modifient la page par cet arbre, à partir duquel est calculé ce que dessine le navigateur.

Le WHATWG maintient ce modèle dans la norme DOM. Son propre résumé cite trois sujets sans les attacher à une plateforme particulière : les arbres de nœuds, les événements et l’annulation d’activités en cours. Le nom est plus ancien que la norme. À l’origine, il désignait une interface pour accéder à un document et le modifier. Aujourd’hui, « le DOM » désigne aussi couramment l’arbre lui-même ; les deux sens s’emploient chaque jour. La norme HTML s’appuie dessus : HTML définit les éléments et leurs fonctions, tandis que la norme DOM définit le nœud, l’élément, l’attribut et l’événement.

L’arbre n’est pas le fichier source. Le HTML d’une page est du texte arrivé par le réseau. Le DOM est le résultat du traitement de ce texte par l’analyseur, qui gère chaque erreur selon les règles de la norme HTML, puis des modifications apportées par tous les scripts exécutés depuis. L’analyseur conserve des détails faciles à oublier. Les espaces entre deux balises peuvent devenir un nœud texte distinct. Le premier enfant d’un élément n’est donc pas toujours le premier élément qu’il contient, même si l’exemple de la norme précise que toute suite d’espaces ne devient pas nécessairement un nœud texte.

Il existe quelques types de nœuds. Le document est à la racine, avec en dessous un doctype, des éléments, des nœuds texte et des commentaires. Un fragment de document est un conteneur léger hors de l’arbre de la page, pratique pour

assembler un bloc avant de l'insérer en une seule opération. Un élément peut aussi héberger un arbre fantôme, un arbre distinct qui lui appartient et que les requêtes ordinaires du document ne parcourent pas.

Les listes de nœuds sont de deux sortes, et la différence compte. Selon la norme, une collection est dynamique sauf indication contraire : `childNodes` et la HTMLCollection de `getElementsByClassName()` changent avec l'arbre. `querySelectorAll()` fait l'objet d'une exception explicite : il renvoie une liste statique, un instantané au moment de l'appel.

La même norme définit le parcours des événements. Un événement est envoyé à un nœud cible et passe par trois phases : la capture, en descendant du sommet de l'arbre par les ancêtres de la cible ; la cible elle-même ; puis la remontée par les ancêtres. La propriété `eventPhase` indique où il se trouve, avec 1, 2 ou 3, et 0 lorsqu'il n'est pas en cours de traitement. Un écouteur ajouté avec `addEventListener()` reçoit des options : `once` le retire après son premier appel, `passive` promet qu'il n'appellera pas `preventDefault()` et `signal` le lie à un `AbortController`, dont l'annulation retire l'écouteur.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que le DOM ? » 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

Tout script modifiant une page passe par le DOM, quel que soit son langage ou son framework. Un framework ajoute sa propre couche, mais le lecteur voit finalement des nœuds de cet arbre. Next.js décrit l'hydratation comme l'attachement par React de gestionnaires d'événements au DOM : le HTML envoyé par le serveur devient ainsi une page qui répond aux clics. Une équipe connaissant l'arbre sous-jacent peut déboguer ce que le framework a réellement fait, plutôt que ce qu'on lui a demandé.

L’arbre est aussi l’endroit où texte et balisage se séparent. Une chaîne affectée à `textContent` remplace les enfants d’un élément par un seul nœud texte et n’est jamais interprétée comme du balisage. La même chaîne affectée à `innerHTML` est remise à l’analyseur HTML. La norme HTML précise que ces accesseurs en écriture ne retirent d’eux-mêmes aucun contenu dangereux. Ce choix décide si le nom d’affichage d’un visiteur est montré comme texte ou exécuté comme code.

L’expérience du lecteur ne dépend pas que de l’arbre. Les styles décident si un nœud est visible, où il se trouve et si un pointeur peut l’atteindre. Rien de cela n’est visible pour un outil qui parcourt seulement les nœuds. C’est dans cet écart que les contrôles automatisés se trompent le plus souvent ; il faut donc tester le rendu d’une page aussi bien que le document sous-jacent.

Enfin, le DOM est le contrat entre une page et tout ce qui la lit après la fin du travail du serveur. Les tests qui cliquent dans la page, les extensions du navigateur et les logiciels d’assistance partent tous de l’arbre dans son état courant. L’autre face du contrat est facile à manquer : un robot qui n’exécute pas JavaScript ne voit que le HTML envoyé, aucun des nœuds ajoutés ensuite par les scripts.

À quoi cela ressemble en pratique

Le script suivant, écrit pour cet article, remplit une liste de commandes depuis les données, conserve un unique écouteur de clic sur la liste plutôt qu’un par élément et peut le retirer ensuite en un appel. `openOrder` représente l’action de la page lorsqu’une commande est choisie.

```
const list = document.querySelector("#orders");
const controller = new AbortController();

function render(orders) {
  const items = orders.map((order) => {
    const li = document.createElement("li");
    li.dataset.id = order.id;
    li.textContent = `${order.number}: ${order.status}`;
    return li;
  });
  list.replaceChildren(...items);
}

list.addEventListener(
  "click",
  (event) => {
```

```
    const li = event.target.closest("li");
    if (li) openOrder(li.dataset.id);
  },
  { signal: controller.signal },
);

// When the view goes away:
controller.abort();
```

Chaque partie correspond à la norme. `createElement()` crée un élément sans parent jusqu'à son insertion. `textContent` place le numéro de commande sous forme de texte : un statut contenant des chevrons reste donc un statut. `replaceChildren()` retire les anciens éléments et insère les nouveaux en une opération. L'écouteur unique sert toute la liste, car le clic sur un élément remonte jusqu'à elle. `closest()` remonte à partir de l'élément cliqué jusqu'au `li` le plus proche. Annuler le contrôleur retire l'écouteur sans imposer à la page de conserver une référence à la fonction.

L'arbre explique aussi une catégorie de défauts révélée par la revue de ce site. Lors d'une revue d'accessibilité menée en août 2026, le site passait ses règles automatisées mais présentait onze défauts, tous liés au rendu plutôt qu'au balisage. Huit liens de pied de page restaient dans l'ordre de tabulation derrière `pointer-events`, une propriété qui ne retire que l'interaction au pointeur. À la lecture du document, ces liens étaient ordinaires ; le pointeur ne les atteignait jamais, mais le clavier s'arrêtait toujours sur chacun. Une animation pilotée par le défilement masquait par rognage le pied de page sur trois pages. Parmi les quatre règles laissées par la revue, l'une met en garde contre la confiance accordée au seul arbre : la visibilité doit être établie d'après les pixels rendus, pas seulement l'arbre du document.

Où les problèmes surviennent

Une boucle sur une collection dynamique saute un élément sur deux. La HTMLCollection de `getElementsByClassName()` est dynamique. Une boucle qui retire la classe de chaque élément par son indice réduit donc sa propre collection et passe un élément sur deux. Parcourez une liste statique : `querySelectorAll()` en renvoie une, et `Array.from()` transforme d'abord une collection dynamique en tableau ordinaire.

```
const pending = document.getElementsByClassName("pending");
for (let i = 0; i < pending.length; i++) {
  pending[i].classList.remove("pending"); // skips every other one
}

for (const el of document.querySelectorAll(".pending")) {
  el.classList.remove("pending"); // static list: all of them
}
```

Les écouteurs s'accumulent. Une vue ajoutant ses écouteurs à chaque rendu sans jamais les retirer exécute chaque gestionnaire une fois par rendu : après trois visites, un clic envoie trois requêtes. Le retrait manuel exige exactement le même objet fonction que l'ajout ; une fonction fléchée écrite directement dans l'appel ne laisse aucune référence à repasser. Transmettez un `signal` issu d'un `AbortController` par vue, comme ci-dessus. Un seul `abort()` retire tous les écouteurs ajoutés par la vue. Utilisez `once: true` pour un écouteur ne devant se déclencher qu'une fois.

Du texte est interprété comme du balisage. Construire une chaîne HTML depuis les données et l'affecter à `innerHTML` remet ces données à l'analyseur sans aucun assainissement par défaut. Un nom d'affichage contenant du balisage, une balise d'image avec gestionnaire d'erreur, par exemple, devient un véritable élément avec son gestionnaire. Le texte va dans `textContent` ou un nœud texte créé par `createTextNode()` . Si du balisage doit réellement venir d'une chaîne, la norme HTML définit `setHTML()` , qui assainit pendant l'analyse et retire le contenu dangereux.

Un observateur s'alimente lui-même. Un `MutationObserver` signale les changements des nœuds surveillés, et sa fonction de rappel s'exécute après chaque lot de changements. Une fonction qui écrit dans le sous-arbre observé pour ajouter un compteur ou corriger une classe crée de nouveaux enregistrements et est rappelée. La page peut alors se bloquer dans sa propre boucle. Ne surveillez que le nécessaire, avec `attributeFilter` et sans `subtree` quand c'est possible. Faites vérifier par la fonction si son changement est déjà en place avant d'écrire, ou appelez `disconnect()` avant l'écriture et `observe()` après.

Pour aller plus loin

- Norme DOM — les définitions WHATWG des nœuds, arbres, collections dynamiques et statiques, événements, `AbortController` et `MutationObserver` .

- Analyse des documents HTML — comment l'analyseur HTML transforme un flux de caractères en objet document, erreurs d'analyse comprises.
- Insertion dynamique de balisage — `innerHTML`, `setHTML()` et `setHTMLUnsafe()`, et lesquels assainissent le contenu.
- Composants serveur et client — l'hydratation selon Next.js : HTML du serveur, puis gestionnaires d'événements attachés au DOM.

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 « Qu'est-ce que le DOM ? » 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/dom/