Engineer

JavaScript

Langage exécuté par les navigateurs

Ce que c’est

JavaScript est un langage de programmation à typage dynamique, normalisé sous le nom ECMAScript, que tout navigateur web peut exécuter et qui fonctionne aussi sur des serveurs et des machines de build grâce à des environnements d’exécution tels que Node.js. On l’abrège en JS. Les éditions annuelles de la norme s’appellent ES2015, ES2024, etc. ; l’ancienne numérotation (ES5, ES6) s’entend encore, et « vanilla JavaScript » désigne le langage sans framework ajouté. Hormis le nom et une partie de la syntaxe à accolades, il a peu de choses en commun avec Java.

Brendan Eich en a écrit la première version chez Netscape, et elle a été distribuée dans le navigateur Navigator 2.0. Ecma International a adopté la première édition de la norme, ECMA-262, en juin 1997, et son comité TC39 en publie une nouvelle chaque année depuis ECMAScript 2016. Une proposition n’entre dans la spécification qu’au stade 4 du processus du comité, qui exige des implémentations déjà distribuées : le texte et les moteurs restent donc rarement en désaccord longtemps.

Le langage dit très peu de choses du monde extérieur. ECMA-262 définit la syntaxe, les types, les objets, les promesses et les modules, et laisse les entrées-sorties à l’environnement qui l’accueille. Dans un navigateur, cet environnement est la plateforme web : le document, les événements, les temporisateurs et les requêtes réseau, spécifiés principalement par le standard HTML et les standards apparentés. Node.js fournit à la place des fichiers, des sockets et des processus. Cette séparation explique une surprise courante : un code peut être du JavaScript parfaitement valide et pourtant cesser de fonctionner en passant d’une page à un serveur, parce que le langage a voyagé mais que son environnement est resté derrière.

Dans un navigateur, les scripts d'une page partagent un seul thread principal avec la mise en page, le dessin et les entrées utilisateur. Une tâche s'exécute jusqu'au bout avant que le navigateur puisse répondre au clic suivant. Les promesses et les fonctions `async` évitent de bloquer ce thread pendant l'attente, tandis que le calcul y reste, sauf s'il est déplacé dans un Web Worker. Les nombres sont des valeurs IEEE 754 à double précision : les entiers ne sont donc exacts que jusqu'à 2^53 − 1, et les valeurs plus grandes nécessitent `BigInt` .

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que JavaScript ? » 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

La véritable décision porte sur la quantité à envoyer aux personnes qui lisent votre site. JavaScript permet à une page de réagir sans aller-retour vers un serveur, et chaque kilooctet est téléchargé, analysé, compilé et exécuté sur un appareil que vous ne contrôlez pas. Un paquet qui paraît instantané sur le portable de la personne qui le développe peut occuper pendant plusieurs secondes un téléphone de milieu de gamme, au prix de la batterie et de la patience du lecteur.

Les scripts sont aussi le point de plus grande exposition d'une page. Le script d'un attaquant, s'il s'exécute, agit avec l'autorité de la page : il peut lire ce que saisit le lecteur et l'envoyer partout où la page est autorisée à envoyer des données. C'est le risque que Content Security Policy vise à contenir. En ne livrant aucun JavaScript exécutable, le site retire ses propres scripts exécutés dans le navigateur de sa surface d'attaque ; les entrées non fiables exigent toujours un traitement sûr et une politique restrictive. L'écosystème des paquets ajoute une seconde exposition. Un processus de construction frontend typique intègre des centaines de dépendances transitives, et chacune est du code qui s'exécute avec les mêmes droits que le vôtre.

L'autre côté du bilan compte aussi. Exécuter le même langage des deux côtés d'un système permet au navigateur et au serveur de partager une règle de validation ou une structure de données, et certains produits ont besoin de beaucoup de scripts : état hors connexion, glisser-déposer, cartes, collaboration en direct. La bonne question se pose pour chaque fonctionnalité : cela doit-il se passer dans le navigateur du lecteur sans interroger un serveur ? Sinon, effectuez le rendu sur le serveur ou au moment de générer les pages, puis livrez du HTML.

Le choix connexe est celui de JavaScript ou TypeScript. TypeScript ajoute un vérificateur de types et une étape de construction ; le code est ensuite compilé en JavaScript ordinaire. Dans une base de code que plusieurs personnes modifient chaque semaine, le vérificateur est généralement rentable ; pour une amélioration de cent lignes sur une page statique, JavaScript et un linter strict suffisent.

À quoi cela ressemble en pratique

Ce site livre à ses lecteurs des pages statiques sans JavaScript exécutable. Son code de recherche est conservé mais désactivé ; JavaScript s'exécute toujours dans les outils de construction et de vérification. Trois pages présentent l'ancienne exception dans le navigateur et le travail réalisé hors du navigateur.

Une version antérieure en trois langues n'avait aucun backend à l'exécution et livrait un unique script propre au site. Le compte rendu de génération avec Hugo décrit cette exception délimitée : un filtre du portfolio dont les commandes restaient cachées jusqu'à l'exécution du script. Ses 108 fichiers de modèles, dont 53 partiels et 18 shortcodes, intervenaient lors de la génération du site. Le compte rendu précise aussi le coût de ce choix à l'époque : les fonctionnalités interactives devaient être prises en charge lors de la génération, et un index de recherche avait fait l'objet d'une estimation de coût, puis avait été mis de côté au lieu d'être livré. Ces chiffres et cette exception décrivent l'implémentation antérieure, pas le code actuellement livré au lecteur.

Le même contenu est publié via Gemini et Gopher, deux protocoles qui n'ont aucune place pour un script. Une génération du site a produit 777 documents dans chaque format à l'étape d'implémentation documentée à partir de la même source que les pages. Une comparaison distincte de versions générées a confirmé qu'aucun octet de HTML n'avait changé lors de l'ajout de ces formats de sortie ; le résultat ne couvre pas les changements éditoriaux ou visuels

ultérieurs. Le site défend l'idée qu'un document n'a pas besoin d'un mégaoctet de scripts pour être lu, et un format qui ne peut en contenir est la manière la plus simple de le montrer.

L'essentiel du JavaScript de ce site se trouve dans l'outillage. À l'étape d'implémentation documentée par la réalisation citée, le contrôle qualité obligatoire du site comprenait trente-deux scripts et 10 242 lignes de JavaScript, alors le plus grand ensemble de code du dépôt. Pendant un temps, c'était le seul ensemble de code qui ne faisait l'objet ni d'un formatage, ni d'une analyse par un linter, ni d'une vérification des types. Le soumettre à un outil de formatage et à un linter a commencé par adapter cet outil au code : le réglage général du dépôt, à quatre espaces, aurait réécrit 2 173 lignes du plus grand vérificateur ; un réglage spécifique à deux espaces pour ces fichiers n'a laissé que 38 lignes présentant de véritables écarts de formatage. Le linter a ensuite signalé 13 problèmes, tous les 13 corrigés sans en masquer aucun. Un bogue dans un vérificateur se manifeste par une vérification qui réussit alors qu'elle devrait échouer : le code du contrôle lui-même reçoit donc le traitement qu'il impose à tout le reste.

Où les problèmes surviennent

Les défaillances ci-dessous découlent de la conception du langage, et chacune a un remède connu.

La page s'affiche, puis ignore les appuis pendant une seconde ou plus. Une tâche longue, comme l'hydratation d'un gros paquet, l'analyse d'une volumineuse réponse JSON ou le tri de milliers de lignes, occupe l'unique thread chargé des entrées. Livrez moins de code pour la première interaction en découpant les paquets par route et en différant le reste ; divisez les travaux longs en morceaux qui rendent la main à la boucle d'événements, et déplacez les calculs lourds dans un Web Worker. Mesurez sur un téléphone lent plutôt que sur le portable où le code a été écrit.

Les prix et les identifiants sont légèrement faux. Tout nombre est une valeur binaire à virgule flottante : les fractions décimales s'arrondissent et les très grands entiers se confondent avec leurs voisins. Une clé de base de données de 64 bits envoyée comme nombre JSON en est la victime habituelle :

```
0.1 + 0.2 === 0.3; // false
Number.MAX_SAFE_INTEGER; // 9007199254740991
9007199254740993 === 9007199254740992; // true
```

Conservez l'argent en unités mineures entières, comme les centimes, envoyez les grands identifiants sous forme de chaînes et utilisez `BigInt` lorsque le calcul doit être exact.

Une comparaison laisse passer la mauvaise valeur. L'opérateur `==` convertit ses opérandes avant de les comparer : `"" == 0` et `"0" == false` sont donc tous deux vrais. Utilisez `===` partout et laissez la règle `eqeqeq` du linter l'imposer, pour que la conversion soit explicitement demandée par le code.

Le texte d'un utilisateur finit par s'exécuter comme du code. Affecter un texte non fiable à `innerHTML`, construire du balisage par concaténation de chaînes ou transmettre une entrée à `eval` transforme des données en script. Écrivez le texte avec `textContent`, envoyez une Content Security Policy qui n'admet les scripts que par nonce ou hachage, et gardez une liste de dépendances courte, aux versions fixées dans un fichier de verrouillage, en lisant ce que chaque mise à jour apporte.

Le code qui fonctionne dans le navigateur plante sur le serveur. Le langage a changé d'endroit, mais pas son environnement : `window` et `document` n'existent que dans un navigateur, et Node.js lit un fichier en CommonJS ou comme module ES selon des réglages que le navigateur ne voit jamais. Gardez les appels propres à un environnement aux frontières du code, définissez explicitement `"type": "module"` dans `package.json` et exécutez chaque partie du code dans l'environnement où elle fonctionnera réellement.

Pour aller plus loin

- Spécification du langage ECMAScript — la version de travail d'ECMA-262, texte implémenté par chaque moteur, actualisé à mesure que les propositions atteignent le stade 4.
- Le processus TC39 — les étapes d'une proposition avant son entrée dans le langage et les exigences de chacune.
- ECMA-262 chez Ecma International — les éditions annuelles publiées, lorsqu'un projet doit nommer la version qu'il cible.

- Standard HTML : boucles d'événements — organisation par le navigateur des tâches, microtâches et du rendu autour des scripts d'une page.
- Standard HTML : l'élément script — scripts classiques et modules, et effet de `async` et `defer` sur le moment de leur exécution.

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 JavaScript ? » 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/javascript/