Node.js intègre enfin le support de TypeScript en natif : avec Node.js v22.18.0 (LTS), les dé… exécuter directement leurs fichiers TypeScript sans configuration supplémentaire

Le 21 août 2025 à 21:13, par Stéphane le calme | 5 commentaires

Node.js intègre enfin le support de TypeScript en natif : avec Node.js v22.18.0 (LTS), les développeurs peuvent enfin exécuter directement leurs fichiers TypeScript sans configuration supplémentaire

Historiquement, le mariage entre Node.js et TypeScript a toujours nécessité une étape de préparation un peu fastidieuse. On devait d'abord compiler le code TypeScript en JavaScript, puis lancer le JavaScript compilé avec Node.js. C'était comme préparer un gâteau en deux fois : d'abord la pâte, puis la cuisson, sans pouvoir sauter l'une des étapes.

Heureusement, les choses ont changé ! Les versions récentes de Node.js, y compris la version 22, peuvent désormais exécuter des fichiers TypeScript directement. C'est un gain de temps considérable, surtout pour les projets simples ou les scripts. On peut maintenant lancer un fichier .ts sans avoir à le transformer au préalable en .js, à condition que le code ne contienne que des éléments qui peuvent être simplement ignorés, comme les annotations de types.

Comment ça marche ?

Node.js gère cette exécution directe grâce à une fonctionnalité appelée "type stripping" (ou "suppression des types"). Au moment de l'exécution, Node.js examine votre code TypeScript et supprime tout ce qui concerne les types, comme les indications string ou number, les laissant se transformer en simples espaces blancs. Le code restant est du JavaScript pur, que Node.js comprend et exécute sans problème.

C'est un peu comme si vous lisiez une recette de cuisine et que vous ignoriez les commentaires sur le type d'ingrédients à utiliser (par exemple, « farine de blé », « sucre blanc »). L'essentiel de la recette (les quantités et les étapes) reste lisible et vous permet de cuisiner sans problème.

Exécuter du TypeScript sans configuration, une petite révolution pour les développeurs

Chaque nouvelle version de Node.js, la célèbre plateforme d'exécution JavaScript, apporte son lot d'améliorations. Mais la sortie de Node.js 22.18.0 LTS (Jod) marque un véritable tournant : pour la première fois, il devient possible d'exécuter directement des fichiers TypeScript sans configuration supplémentaire ni outil tiers. Une avancée technique qui peut sembler discrète, mais qui change la vie de nombreux développeurs et simplifie l'adoption de TypeScript, devenu un standard incontournable.

Qu'est-ce que Node.js, au juste ?

Pour bien comprendre la portée de cette nouveauté, faisons un petit détour. Node.js est une plateforme qui permet d'exécuter du JavaScript en dehors du navigateur. Concrètement, c'est grâce à Node.js que l'on peut utiliser JavaScript pour créer des serveurs web, des applications back-end, des outils en ligne de commande, et même des applications multiplateformes.

Depuis sa création en 2009, Node.js est devenu l'un des piliers du développement moderne. On lui doit notamment la montée en puissance du JavaScript au-delà du simple « langage du web ».

Et TypeScript, dans tout ça ?

TypeScript, créé par Microsoft en 2012, est une surcouche de JavaScript qui ajoute un système de types. Cela signifie que les développeurs peuvent préciser dans leur code à quoi sert chaque variable, fonction ou objet. Cela leur permet de détecter les erreurs potentielles pendant la phase de développement plutôt qu'au moment de l'exécution.

Les caractéristiques de TypeScript, telles que les interfaces, les espaces de noms et l'inférence de type, en font une option attrayante pour les applications à grande échelle où un code propre et fiable est primordial. Alors que JavaScript devient de plus en plus complexe avec la programmation asynchrone et de nombreuses bibliothèques, TypeScript peut servir de filet de sécurité pour les développeurs.

TypeScript, qui, comme son nom l'indique, est un langage fortement typé, se compile en JavaScript et est populaire pour les applications qui s'exécutent soit dans le navigateur, soit sur un moteur d'exécution JavaScript tel que V8, parrainé par Google et utilisé par Node.js. Les enquêtes placent généralement TypeScript parmi les dix premiers langages de programmation. Par exemple, dans l'édition de juillet 2025 de l'index PYPL qui mesure la popularité des langages en fonction de la fréquence des recherches de tutoriels pour chaque langage sur Google, TypeScript est classé à la 7ème position.

D'ailleurs, Robert Vitonsky est allé plus loin en déclarant : « refuser TypeScript est un signal que vous ne vous souciez pas de la qualité du code ».

Et d'estimer que :

> ***Robert Vitonsky***
> Le contrôle de la qualité du code est un processus complexe qui permet de maintenir le code à jour. Vous ne pouvez pas simplement couvrir le code avec des tests à 100% ou examiner chaque pull request et être sûr que votre code est maintenable, et quelqu'un d'autre que vous peut le découvrir dans ce désordre.
>
> Vous ne pouvez pas vous assurer que votre code n'a pas de bogues et qu'il est parfaitement maintenable. Vous pouvez seulement augmenter les structures défensives dans votre référentiel pour rendre difficile la diffusion de mauvais code avec des bogues. Plus il y a de barrières au mauvais code, meilleure est la qualité du code.
>
> Cela signifie que vous devez utiliser toutes les méthodes ensemble pour protéger le code dans votre référentiel : tests unitaires/e2e/intégration, revue de code, outils d'analyse de code, et maintenir une documentation claire, etc.
>
> TypeScript est un outil d'analyse de code puissant ; il peut détecter de nombreux défauts dans le code. Un compilateur TypeScript oblige les programmeurs à s'assurer que le code est correct au niveau des types. La valeur du typage statique est sous-estimée par David et beaucoup d'autres.

La nature dynamique de JavaScript peut le rendre plus sujet aux erreurs et peu sûr, alors que le compilateur TypeScript détecte plus d'erreurs de codage avant le déploiement - mais seulement si le contrôle de type est effectué, ce que le support léger de Node.js ne fait pas.

Node.js a été créé par Ryan Dahl en tant que moteur d'exécution JavaScript et publié pour la première fois en 2009. Il reste de loin le moteur d'exécution JavaScript le plus populaire, bien que Dahl lui-même ait créé un second projet appelé Deno (qui, selon lui, corrige les « erreurs de conception de Node »). Deno utilise TypeScript comme langage principal et a eu un certain impact. Tout comme bun - un autre runtime qui « traite TypeScript comme un citoyen de première classe » bien qu'il ne vérifie pas le code. Deno, en revanche, peut vérifier le typeScript mais ne le fait pas par défaut pour des raisons de performance.

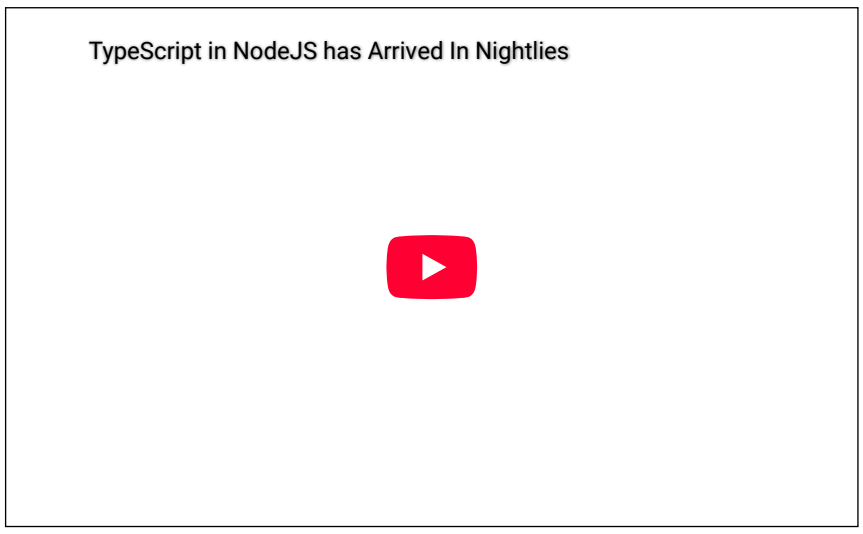
Par exemple :

Code TypeScript : Sélectionner tout

```
function greet(name: string): string {
  return `Bonjour, ${name}`;
}
```

Ici, le `: string` indique que la variable `name` doit obligatoirement être une chaîne de caractères. Résultat : le code est plus clair, plus robuste, et les erreurs sont détectées plus tôt.

Le revers de la médaille : TypeScript doit être compilé (traduit) en JavaScript pour fonctionner. Jusqu'à présent, on ne pouvait pas simplement lancer node fichier.ts : il fallait d'abord passer par un compilateur (tsc) ou un outil tiers (ts-node, tsx).



Ce qui change avec Node.js 22.18.0

Désormais, plus besoin d'étape intermédiaire : Node.js sait lire directement les fichiers .ts.

Le type stripping automatique

Node.js retire simplement les annotations de types (`: string`, `: number`, etc.) et transforme le code en JavaScript exécutable. C'est ce que l'on appelle le type stripping.

Par exemple :

Code Bash : Sélectionner tout

```
$ echo 'const foo: string = "World"; console.log(`Hello ${foo}!`);' > file.ts
$ node file.ts
Hello World!
```

En un seul appel, sans configuration ni outil externe, le script TypeScript s'exécute.

Mais avec des limites

Certaines fonctionnalités avancées de TypeScript ne sont pas encore prises en charge. Par exemple :

- les enum (enum Couleur { Rouge, Vert, Bleu }),
- les namespace contenant du code,
- les paramètres d'accès dans les constructeurs (constructor(private name: string)),
- et d'autres cas spécifiques.

Pour gérer ces situations, Node.js propose une option supplémentaire : `--experimental-transform-types`, qui active une transformation plus complète, mais encore expérimentale.

Ce que cela change dans la pratique

Pour les développeurs débutants

C'est une excellente nouvelle ! Vous pouvez écrire un script TypeScript simple et le lancer immédiatement, sans installer de compilateur, sans apprendre de configuration complexe. Cela réduit la barrière d'entrée et encourage l'adoption de TypeScript par les nouveaux venus.

Pour les projets rapides ou les prototypes

Besoin de tester une idée en quelques minutes ? Vous pouvez écrire en TypeScript, profiter des avantages des types, et lancer directement le fichier. Cela accélère énormément les phases d'expérimentation.

Pour les grandes entreprises et projets complexes

Attention, ici la nuance est importante : Node.js ne remplace pas complètement l'écosystème existant. Les entreprises qui utilisent TypeScript à grande échelle continueront probablement d'employer des outils comme ts-node, Babel ou esbuild, car ils offrent plus de contrôle (transpilation avancée, optimisation des performances, gestion de tsconfig.json, etc.).

Et la communauté dans tout ça ?

Cette évolution a suscité beaucoup de réactions positives. Depuis longtemps, la communauté demandait un support natif de TypeScript dans Node.js, car le langage est désormais utilisé massivement : selon GitHub, plus de 40 % des projets modernes utilisent TypeScript.

Cependant, certains développeurs restent prudents :

- L'implémentation actuelle est expérimentale et pourrait évoluer.
- Elle ne remplace pas un compilateur complet.
- Le risque est que certains développeurs se lancent en pensant pouvoir tout faire directement avec Node.js, et se heurtent vite aux limites.

Une étape vers une convergence JavaScript – TypeScript ?

Cette nouveauté illustre une tendance plus large : la frontière entre JavaScript et TypeScript devient de plus en plus fine.

- JavaScript reste le langage d'exécution : tout code doit finir traduit en JavaScript pour tourner.
- TypeScript devient la norme d'écriture : de plus en plus de développeurs ne veulent plus coder sans les annotations de types.

En intégrant nativement le support minimal de TypeScript, Node.js reconnaît cette réalité et facilite la transition.

Conclusion

L'arrivée du support natif de TypeScript dans Node.js 22.18.0 est une avancée qui pouvait déjà être anticipée, notamment lorsque Node.js a ajouté un support expérimental pour TypeScript. Elle simplifie la vie des développeurs, démocratise encore plus TypeScript et ouvre la voie à un futur où la configuration sera réduite au minimum.

Néanmoins, ce n'est pas (encore) une solution miracle : pour les projets sérieux et complexes, les outils spécialisés restent indispensables. Mais pour les scripts, les prototypes ou l'apprentissage, cette nouveauté est une petite révolution qui montre à quel point JavaScript et TypeScript sont devenus indissociables.

Source : note de version Node.js v22.18.0 (LTS)

Et vous ?

➡ Pensez-vous que l'exécution directe de fichiers TypeScript dans Node.js est une véritable révolution ou juste un gadget pratique ?