Engineer

TypeScript

JavaScript із типами, перевіреними до запуску

Що це таке

TypeScript — мова програмування від Microsoft, яка додає до JavaScript анотації типів, разом із компілятором `tsc`, що їх перевіряє. Назву скорочують до TS, а файли мають розширення `.ts` або `.tsx`, коли містять розмітку JSX. Синтаксис JavaScript є коректним TypeScript, тому мову називають надмножиною JavaScript. Компілятор відкритий під ліцензією Apache License 2.0.

Типи живуть лише доти, доки триває перевірка. Звичайні анотації типів стираються, коли TypeScript генерує JavaScript; вони не перевіряють дані під час роботи програми й не потребують бібліотеки для перевірки типів під час виконання. Анотації типів не змінюють поведінки коду JavaScript під час роботи. Цей один факт пояснює і силу мови, і її межі: засіб перевірки аналізує типи, описані у вихідному коді; він не перевіряє дані, які надходять уже під час роботи програми.

Система типів структурна. Два типи сумісні, коли збігається їхня форма, хоч би як вони називалися, і це пасує звичці JavaScript передавати звичайні об’єкти; C# і Java натомість порівнюють оголошені імена. Більшу частину типів компілятор виводить сам, тож анотації стоять переважно на сигнатурах функцій та експортованих значеннях. У системі також є кілька навмисних прогалин. `any` вимикає перевірку для значення, а твердження типу через `as` просить компілятор повірити програмістові на слово. Обидва зникають із результату разом з усім іншим.

Є три поширені способи запуску. `tsc` перевіряє проєкт і генерує JavaScript. Бандлери на кшталт esbuild вилучають типи без перевірки й лишають її окремому запуску `tsc --noEmit`. Node.js, починаючи з v23.6.0 і v22.18.0, виконує файли `.ts` напряму, замінюючи синтаксис типів пробілами, знову

ж таки без перевірки типів, і приймає лише той синтаксис, який можна так прибрати. TypeScript 7.0, анонсований у липні 2026 року, — це компілятор, перенесений на Go, який, за словами команди, зазвичай від 8 до 12 разів швидший на повних збираннях.

Застосуйте поняття до рішення

Застосуйте «Що таке TypeScript?» 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

Питання, на яке відповідає команда: чи окупають крок збирання й перевірка своє місце тим, що ловлять помилки до запуску коду. У кодовій базі, яку щотижня змінюють кілька людей, зазвичай окупають. Перейменування поля показує кожне місце, де воно використовувалося, редактор доповнює й пояснює код на основі типів, а сигнатура модуля стає контрактом, який викликачі не можуть тихо порушити. Ціна — перевірка в кожному конвеєрі, файл конфігурації з безліччю параметрів і визначення типів, які мають встигати за бібліотеками, що їх описують.

Найважливіше налаштування — суворість. Прапорець `strict` вмикає родину перевірок, серед них `strictNullChecks` і `noImplicitAny`, і посібник порівнює результат із регулятором, а не з перемикачем. Починаючи з TypeScript 6.0, цей прапорець увімкнений за замовчуванням, тож новий проєкт одразу стартує суворим. Старіший проєкт може досі мати значення false, і якщо ввімкнути прапорець пізніше, доведеться пройтися по кожному файлу, якого це стосується. Дві корисні перевірки стоять поза цією родиною і вмикаються окремо: `noUncheckedIndexedAccess` та `exactOptionalPropertyTypes`.

Стирання типів визначає межу. Програма, яка читає JSON із мережевого запиту, файлу, черги чи форми, має щодо форми цих даних лише слово програміста, бо коли дані надходять, типів уже немає. Тому межі з зовнішнім світом потребують валідації під час виконання, найкраще згенерованої з тієї самої схеми, що й типи, щоб перевірка і тип завжди збігалися. TypeScript робить узгодженим нутро програми, а краї залишає коду.

Перевірка типів живить і інші інструменти. Лінтер на кшталт typescript-eslint може запитати в компілятора інформацію про типи, і тоді правило здатне позначити проміс, залишений без `await`, або значення типу `any`, що розповзається модулем. Так команда перетворює власні домовленості на помилки, а не на коментарі в рев'ю.

Як це виглядає на практиці

У проєктах, показаних на цьому сайті, TypeScript тримають чесним згенерований код та інструменти, а не домовленості. Три сторінки показують, як саме.

На одній платформі TypeScript-клієнт фронтенду ніколи не пишуть вручну. Його генерують з опису самого API, який Huma формує з типів Go, а ті своєю чергою випливають зі схеми бази даних та її функцій. На задокументованому етапі реалізації клієнт налічував 44 076 рядків, а 61 mock-обробник, згенерований поруч із ним, давав тестам фронтенду працювати з реальним контрактом замість рукописної фікстури. Кожен перехід має свій запобіжник. Перевірка контракту в CI порівнює те, що надсилає фронтенд, із тим, що очікує API, і валить білд за розбіжності, а під нею працює перевірка схеми, яка звіряє реальну структуру даних. Вона охоплює опціональні поля тіла запиту, де легко сплутати «відсутнє» і null, та enum-и query-параметрів. Поле, змінене з одного боку, тепер валить білд за кілька хвилин після зміни, а не виринає як 422 у браузері вже після того, як розбіжність потрапила в продакшн. Натомість зміна контракту доходить до типів клієнта лише після повторного запуску всього ланцюжка, а повторна генерація потребує зайвих зусиль.

Інша сторінка задає одну планку для шести мов, серед яких і TypeScript. Кожна діагностика ESLint вважається помилкою, рівня попереджень, у якому можна сховатися, там немає, а inline-придушення заборонені, тож знахідку доводиться виправляти, а не глушити. Pre-commit- і pre-push-хуки запускають лінтери й тести, а обмеження на довжину функцій і файлів тримають модулі читабельними.

Лінтери, що постачаються з мовою, не дістають до правил, специфічних для проєкту, тому п'ятнадцять власних правил ESLint утримували прийняті в цьому репозиторії патерни TypeScript на задокументованому етапі реалізації. Вони належали до рушія guard-перевірок із 274 скриптів, 268 з яких були зареєстровані в commit-хуках, а самі перевірки мали 146 власних тестів:

зламана перевірка пропускає все, і ніхто цього не помічає. Одне правило стосується тестів, а не типів: коміт із префіксом `fix:` мусить містити тест, який без виправлення падає. Час рев'ю перемістився з механіки на проєктування, а натомість комітити стало повільно.

Де виникають проблеми

Більшість збоїв у TypeScript виникає тоді, коли типам довіряють далі, ніж бачить компілятор.

Поле з типом string під час виконання виявляється undefined. Дані прийшли ззовні, і їм дали тип замість перевірки. Бібліотека DOM типізує `res.json()` як проміс від `any`, тож компілятор приймає все, що код робить із результатом, а твердження типу робить довіру явною, але не безпечною:

```
const res = await fetch("/api/user");
const user = (await res.json()) as User; // trusted, never checked
user.email.toLowerCase(); // TypeError if email is missing
```

Зовнішні дані варто обробляти як `unknown`, валідувати на межі за схемою часу виконання й генерувати тип із тієї самої схеми, щоб вони не розходилися. Правило лінтера з інформацією про типи тоді позначить кожен `any`, що прослизне.

Код із помилками типів однаково потрапляє в продакшн. `tsc` записує JavaScript навіть тоді, коли повідомляє про помилки, бо `noEmitOnError` за замовчуванням має значення false, а бандлери й Node.js вилучають типи взагалі без перевірки. Варто запускати `tsc --noEmit` як обов'язковий крок у CI та в pre-commit-хуку й валити білд за його кодом виходу.

Звернення за індексом повертає undefined, який тип виключав. Елемент масиву або ключ, прочитаний через індексну сигнатуру, отримує тип так, ніби значення є завжди, навіть за кінцем масиву. Допомагає `noUncheckedIndexedAccess`: він додає `undefined` до таких читань і змушує код це обробити. У парі з ним варто ввімкнути `exactOptionalPropertyTypes`, який відрізняє відсутню властивість від властивості зі значенням `undefined`.

Файл, що компілюється через tsc, падає, коли його запускає Node.js. Вилучення типів прибирає лише синтаксис, що не впливає на виконання. Enum-и та простори імен зі значеннями зупиняють Node.js із помилкою `ERR_UNSUPPORTED_TYPESCRIPT_SYNTAX`, а тип, імпортований без ключового слова `type`, вважається імпортом значення й падає під час виконання. Прапорці `erasableSyntaxOnly` і `verbatimModuleSyntax` змушують компілятор повідомити про обидва випадки раніше за Node.js; для типів варто писати `import type`, а enum-и замінювати об'єктними літералами з позначкою `as const`.

Що почитати далі

- TypeScript Documentation — посібник, довідкові сторінки й примітки до випусків, які підтримує команда TypeScript.
- The Basics — розділ посібника про статичну перевірку, генерацію коду попри помилки, стерті типи й суворість.
- TSConfig Reference: strict — усі параметри компілятора з їхніми типовими значеннями, зокрема перевірки, які вмикає прапорець strict.
- Type Compatibility — структурна підтипізація та місця, де система типів навмисно ненадійна.
- Node.js: Modules: TypeScript — вилучення типів, синтаксис, який воно відхиляє, і налаштування компілятора, які радить Node.js.
- Linting with Type Information — як typescript-eslint дає правилам лінтера доступ до перевірки типів.

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 TypeScript?» 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/typescript/