Engineer

TypeScript

JavaScript med typer tjekket før det kører

Hvad det er

TypeScript er et programmeringssprog fra Microsoft, der føjer typeannotationer til JavaScript, sammen med en compiler, `tsc`, der tjekker dem. Navnet forkortes til TS, og filerne ender på `.ts`, eller `.tsx` når de indeholder JSX-markup. JavaScript-syntaks er gyldig TypeScript, og derfor kaldes sproget et supersæt af JavaScript. Compileren er open source under Apache License 2.0.

Typerne lever kun, så længe tjekket står på. Almindelige typeannotationer slettes, når TypeScript genererer JavaScript; de validerer ikke data under kørslen og kræver ikke noget bibliotek til typekontrol under kørslen. Typeannotationer ændrer ikke, hvordan JavaScript-koden opfører sig, når den kører. Den ene kendsgerning forklarer både styrken og grænsen: tjekkeren ræsonnerer om de typer, der er beskrevet i kildekoden; den validerer ikke data, der ankommer, mens programmet kører.

Typesystemet er strukturelt. To typer er forenelige, når deres form passer, uanset hvad de hedder, og det passer godt til JavaScripts vane med at sende almindelige objekter rundt; C# og Java sammenligner i stedet de erklærede navne. Meget af typningen udledes automatisk, så annotationerne sidder mest på funktionssignaturer og eksporterede værdier. Systemet har også et par bevidste huller. `any` slår tjekket fra for en værdi, og en assertion skrevet med `as` beder compileren tage programmørens ord for en type. Begge forsvinder fra outputtet sammen med alt det andet.

Der er tre almindelige måder at køre det på. `tsc` tjekker et projekt og udsender JavaScript. Bundlere som esbuild fjerner typerne uden at tjekke dem og overlader tjekket til en separat kørsel af `tsc --noEmit`. Node.js kører siden v23.6.0 og v22.18.0 `.ts`-filer direkte ved at erstatte typesyntaksen med

mellemrum, igen uden typetjek, og accepterer kun syntaks, der kan fjernes på den måde. TypeScript 7.0, annonceret i juli 2026, er en portering af compileren til Go, som holdet bag oplyser typisk er 8 til 12 gange hurtigere ved fulde builds.

Brug begrebet i en beslutning

Brug »Hvad er TypeScript?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Brugerflader, der indlæses hurtigt, virker på en telefon og opfører sig ens i alle de browsere, jeres kunder nu bruger.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har styret full-stack-udviklingen af GIS-kort — full stack-kortløsning, fra database til browser

Har styret full-stack-udviklingen af GIS-kort Kontakt os →

Hvorfor det betyder noget

Spørgsmålet, et team skal besvare, er, om et byggetrin og en tjekker gør sig fortjent til pladsen ved at fange fejl, før koden kører. I en kodebase, som flere mennesker ændrer hver uge, gør de som regel. Omdøber man et felt, viser compileren hvert sted, der brugte det, editoren fuldfører og forklarer kode ud fra typerne, og et moduls signatur bliver en kontrakt, som kalderne ikke i det stille kan bryde. Prisen er et tjek i hver byggepipeline, en konfigurationsfil med mange indstillinger og typedefinitioner, der skal følge med de biblioteker, de beskriver.

Strenghed er den indstilling, der betyder mest. Flaget `strict` slår en familie af tjek til, blandt andre `strictNullChecks` og `noImplicitAny`, og håndbogen beskriver resultatet som en drejeknap snarere end en kontakt. Siden TypeScript 6.0 er flaget slået til som standard, så et nyt projekt starter strengt. Et ældre projekt kan stadig have det sat til false, og slår man det til senere, skal hver berørt fil igennem. To nyttige tjek står uden for familien og skal slås til hver for sig: `noUncheckedIndexedAccess` og `exactOptionalPropertyTypes`.

Sletningen af typerne sætter grænsen. Et program, der læser JSON fra et netværkskald, en fil, en kø eller en formular, har kun programmørens ord for, hvilken form dataene har, for typerne er væk, når dataene ankommer. Grænserne til omverdenen har derfor brug for validering under kørslen, helst genereret fra det samme skema som typerne, så tjekket og typen aldrig kan blive uenige. TypeScript gør programmets indre konsistent og overlader kanterne til koden.

Tjekkeren fodrer også andre værktøjer. En linter som typescript-eslint kan bede compileren om typeinformation, så en regel kan markere et promise, som ingen venter på, eller en værdi af typen `any`, der breder sig gennem et modul. Sådan gør et team sine egne konventioner til fejl i stedet for kommentarer i et review.

Sådan ser det ud i praksis

I de projekter, der vises på dette site, holdes TypeScript ærligt af genereret kode og af værktøjer frem for af konventioner. Tre sider viser hvordan.

På én platform bliver frontendens TypeScript-klient aldrig skrevet i hånden. Den genereres ud fra API'ets egen beskrivelse, som Huma udsender ud fra Go-typerne, der igen følger databaseskemaet og dets funktioner. På det dokumenterede implementeringstrin fyldte klienten 44.076 linjer, og 61 mock-handlere, der blev genereret ved siden af den, lod frontendens tests køre mod den rigtige kontrakt i stedet for en håndskrevet fixture. Hvert hop har sit eget værn. Et kontrakttjek i CI sammenligner det, frontenden sender, med det, API'et forventer, og får buildet til at fejle, når de afviger, med en schema-probe nedenunder, der undersøger de rigtige former. Det dækker valgfrie body-felter, hvor »mangler« og »null« let forveksles, og enums i query-parametre. Et felt, der ændrer sig på den ene side, får nu et build til at fejle minutter efter ændringen, i stedet for at dukke op som et 422 i browseren, efter at uoverensstemmelsen er udgivet. Til gengæld når en ændring af kontrakten kun frem til klientens typer ved at køre hele kæden igen, og det er besværligt at regenerere.

En anden side sætter én standard for seks sprog, heriblandt TypeScript. Hver diagnostik fra ESLint tæller som en fejl, der findes intet advarselsniveau at gemme sig i, og inline-suppressions er forbudt, så et fund skal rettes i stedet for at blive gjort tavst. Pre-commit- og pre-push-hooks kører linterne og testene, og grænser for funktions- og fillængde holder modulerne læsbare.

De lintere, der følger med et sprog, når ikke frem til et projekts egne regler, så femten specialskrevne ESLint-regler holdt repoets egne kodemønstre på plads i TypeScript på det dokumenterede implementeringstrin. De hørte til en guard-motor med 274 check-scripts, hvoraf 268 var registreret i commit-hooks, og guard-laget havde 146 test for sig selv, fordi en ødelagt guard lader alt passere, uden at nogen opdager det. Én regel handler om test snarere end typer: en commit med præfikset `fix:` skal have en test med, der fejler uden rettelsen. Review-tiden flyttede fra mekanik til design, og til gengæld blev det langsomt at committe.

Hvor det går galt

De fleste fejl i TypeScript opstår, fordi man stoler på typerne længere ud, end compileren kan se.

Et felt med typen string viser sig at være undefined under kørslen. Dataene kom udefra og fik en type i stedet for et tjek. DOM-biblioteket giver `res.json()` typen promise af `any`, så compileren accepterer alt, hvad koden gør med resultatet, og en assertion gør tilliden eksplicit uden at gøre den sikker:

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

Behandl data udefra som `unknown`, valider dem ved grænsen mod et runtime-skema, og generer typen fra det samme skema, så de to følges ad. En lintregel med typeinformation kan derefter markere ethvert `any`, der slipper igennem.

Kode med typefejl når alligevel i produktion. `tsc` skriver stadig JavaScript, når den melder fejl, fordi `noEmitOnError` som standard er false, og bundlere og Node.js fjerner typer uden overhovedet at tjekke dem. Kør `tsc --noEmit` som et obligatorisk trin i CI og i pre-commit-hooken, og lad buildet fejle på dets exitkode.

Et opslag via indeks giver undefined, som typen havde udelukket. Et array-element eller en nøgle, der læses gennem en indekssignatur, får en type, som om værdien altid er der, også forbi arrayets sidste element. Slå `noUncheckedIndexedAccess` til, som føjer `undefined` til den slags læsninger og tvinger koden til at håndtere det. Kombinér det med `exactOptionalPropertyTypes`, der skelner mellem en egenskab, der mangler, og en, der er sat til `undefined`.

En fil, der kompilerer med tsc, fejler, når Node.js kører den. Type stripping fjerner kun syntaks, der er uden betydning under kørslen. Enums og namespaces med værdier får Node.js til at stoppe med `ERR_UNSUPPORTED_TYPESCRIPT_SYNTAX`, og en type, der importeres uden nøgleordet `type`, behandles som en værdiimport, der fejler under kørslen. Slå `erasableSyntaxOnly` og `verbatimModuleSyntax` til, så compileren melder begge dele, før Node.js gør det, skriv `import type` for typer, og erstat enums med objektliteraler markeret `as const`.

Læs mere

- TypeScript Documentation — håndbogen, referencesiderne og release notes, vedligeholdt af TypeScript-holdet.
- The Basics — håndbogens kapitel om statisk tjek, output trods fejl, slettede typer og strenghed.
- TSConfig Reference: strict — alle compilerindstillinger med deres standardværdi, også de tjek, flaget strict slår til.
- Type Compatibility — strukturel subtypning og de steder, hvor typesystemet har huller med vilje.
- Node.js: Modules: TypeScript — type stripping, den syntaks, den afviser, og de compilerindstillinger, Node.js anbefaler.
- Linting with Type Information — hvordan typescript-eslint giver lintregler adgang til typetjekkeren.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er TypeScript?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/typescript/