Engineer

TypeScript

JavaScript med typer som kontrolleras innan det körs

Vad det är

Microsoft står bakom TypeScript, ett programspråk som lägger till typannoteringar i JavaScript och levereras med en kompilator, `tsc`, som kontrollerar dem. Namnet förkortas ofta TS, och filerna slutar på `.ts`, eller på `.tsx` när de innehåller JSX-markering. All syntax i JavaScript är giltig TypeScript, och därför kallas språket en övermängd (superset) av JavaScript. Kompilatorn är öppen källkod under Apache License 2.0.

Typerna lever bara så länge kontrollen pågår. Vanliga typannoteringar raderas när TypeScript genererar JavaScript; de validerar inte data vid körning och kräver inget bibliotek för typkontroll vid körning. Typannoteringar ändrar inte hur JavaScript-koden beter sig när den körs. Detta enda faktum förklarar både värdet och gränserna: typkontrollen resonerar om typerna som beskrivs i källkoden; den validerar inte data som kommer in medan programmet körs.

Typsystemet är strukturellt. Två typer är kompatibla när deras form stämmer, oavsett vad de heter, och det passar vanan i JavaScript att skicka runt vanliga objekt. C# och Java jämför i stället deklarerade namn. En stor del av typerna härleds, så annoteringarna sitter mest på funktionssignaturer och exporterade värden. Systemet är också med avsikt osunt (unsound) på några ställen. `any` stänger av kontrollen för ett värde, och en typförsäkran skriven med `as` säger åt kompilatorn att lita på programmerarens ord om en typ. Båda försvinner ur utdata tillsammans med allt annat.

Det finns tre vanliga sätt att köra språket. `tsc` kontrollerar ett projekt och genererar JavaScript. Paketerare (bundlers) som esbuild skalar bort typerna utan att kontrollera dem och lämnar kontrollen åt en separat körning av `tsc --noEmit`. Node.js kör sedan v23.6.0 och v22.18.0 `.ts`-filer direkt genom att ersätta typsyntaxen med blanksteg, även det utan typkontroll, och godtar bara syntax

som går att radera på det sättet. TypeScript 7.0, som tillkännagavs i juli 2026, är en portering av kompilatorn till Go, och teamet uppger att den vanligtvis är 8–12 gånger snabbare vid fullständiga byggen.

Använd begreppet i ett beslut

Använd ”Vad är TypeScript?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Gränssnitt som laddar snabbt, fungerar i mobilen och beter sig likadant i alla webbläsare dina kunder råkar använda.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Styrde full stack-utvecklingen av GIS-kartor — kartutveckling i full stack, från databas till webbläsare

Styrde full stack-utvecklingen av GIS-kartor Hör av er →

Varför det spelar roll

Frågan ett team besvarar är om ett byggsteg och en typkontroll gör sig förtjänta av sin plats genom att fånga fel innan koden körs. I en kodbas som flera personer ändrar varje vecka gör de oftast det. Byter du namn på ett fält visas varje ställe som använde det, editorn kompletterar och förklarar kod utifrån typerna, och signaturen för en modul blir ett kontrakt som anroparna inte kan bryta i tysthet. Kostnaden är en kontroll i varje pipeline, en konfigurationsfil med många alternativ och typdefinitioner som måste hålla jämna steg med biblioteken de beskriver.

Striktheten är den inställning som betyder mest. Flaggan `strict` slår på en hel familj av kontroller, bl.a. `strictNullChecks` och `noImplicitAny`, och handboken beskriver resultatet som en ratt snarare än en strömbrytare. Sedan TypeScript 6.0 är flaggan på som standard, så ett nytt projekt börjar strikt. Ett äldre kan fortfarande sätta den till false, och att slå på den i efterhand innebär att gå igenom varje fil den berör. Två nyttiga kontroller ligger utanför familjen och måste slås på var för sig: `noUncheckedIndexedAccess` och `exactOptionalPropertyTypes`.

Raderingen sätter gränsen. Ett program som läser JSON från en nätverksförfrågan, en fil, en kö eller ett formulär har bara programmerarens ord för hur de data ser ut, eftersom typerna är borta när data kommer in. Gränserna behöver därför validering vid körning, helst genererad från samma schema som typerna, så att kontrollen och typen inte kan säga emot varandra. TypeScript gör insidan av ett program konsekvent och lämnar kanterna åt koden.

Typkontrollen matar också andra verktyg. En linter som typescript-eslint kan be kompilatorn om typinformation, så att en regel kan flagga ett promise som ingen väntar in eller ett värde av typen `any` som sprider sig genom en modul. Så gör ett team sina egna konventioner till fel i stället för till kommentarer i granskningen.

Så ser det ut i praktiken

I projekten som visas på den här webbplatsen hålls TypeScript ärligt av genererad kod och av verktyg snarare än av konvention. Tre sidor visar hur.

På en plattform skrivs TypeScript-klienten för frontenden aldrig för hand. Den genereras från API:ets egen beskrivning, som Huma tar fram ur Go-typerna, som i sin tur följer databasschemat och dess funktioner. I det dokumenterade implementeringsskedet var klienten på 44 076 rader, och 61 mock-hanterare som genererades bredvid den lät frontendens tester köras mot det verkliga kontraktet i stället för mot en handskriven fixture. Varje led har sitt eget skydd. En kontraktskontroll i CI jämför vad frontenden skickar med vad API:et förväntar sig och stoppar bygget när de skiljer sig åt, med en schemasond under som undersöker de verkliga strukturerna. Den täcker valfria fält i anropets body, där ”saknas” och ”null” lätt blandas ihop, och enum-värden i query-parametrar. Ett fält som ändras på ena sidan fäller nu ett bygge minuter efter ändringen, i stället för att dyka upp som en 422 i webbläsaren när avvikelsen redan har släppts. I gengäld når en ändring i kontraktet klientens typer först när hela kedjan körs igen, och det är omständligt att generera om.

En annan sida sätter en och samma ribba för sex språk, och TypeScript är ett av dem. Varje diagnostikmeddelande som ESLint rapporterar är ett fel, det finns ingen varningsnivå att gömma sig i, och undertryckningar direkt i koden är förbjudna, så ett fynd måste åtgärdas i stället för att tystas. Hookar för pre-commit och pre-push kör linters och tester, och gränser för längden på funktioner och filer håller modulerna läsbara.

De lintverktyg som följer med ett språk stannar före ett projekts egna regler, så femton egna ESLint-regler upprätthöll repots etablerade mönster i TypeScript i det dokumenterade implementeringsskedet. Reglerna ingick i en vaktmotor med 274 kontrollskript, varav 268 var registrerade i commit-hookarna, och vakterna hade 146 egna tester, eftersom en trasig vakt släpper igenom allt utan att någon märker det. En regel gäller tester snarare än typer: en commit med prefixet

`fix:` måste innehålla ett test som fallerar utan rättelsen. Tiden i kodgranskningen flyttades från mekanik till design, och i gengäld gick det långsamt att committa.

Var det går fel

De flesta fel med TypeScript kommer av att man litar på typerna längre än kompilatorn kan se.

Ett fält som är typat som sträng visar sig vara undefined vid körning. Data kom utifrån programmet och fick en typ i stället för att kontrolleras. DOM-biblioteket typar `res.json()` som ett promise av `any`, så kompilatorn godtar vad koden än gör med resultatet, och en typförsäkran gör förtroendet uttryckligt utan att göra det säkert:

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

Behandla data utifrån som `unknown`, validera dem vid gränsen med ett schema som körs, och generera typen från det schemat så att de två håller jämna steg. En typmedveten lintregel kan sedan flagga varje `any` som slinker igenom.

Kod med typfel når produktion ändå. `tsc` skriver fortfarande JavaScript när den rapporterar fel, eftersom `noEmitOnError` som standard är false, och paketerare och Node.js skalar bort typerna utan att kontrollera dem alls. Kör `tsc --noEmit` som ett obligatoriskt steg i CI och i pre-commit-hooken, och låt bygget fallera på dess slutkod.

En indexuppslagning ger undefined, fast typen uteslöt det. Ett element i en array, eller en nyckel som läses via en indexsignatur, typas som alltid närvarande, även förbi arrayens slut. Slå på `noUncheckedIndexedAccess`, som lägger till `undefined` i de läsningarna och tvingar koden att hantera det. Kombinera den med `exactOptionalPropertyTypes`, som skiljer en egenskap som saknas från en som är satt till `undefined`.

En fil som kompilerar med tsc stannar när Node.js kör den. Att skala bort typer tar bara bort syntax som saknar betydelse vid körning. Enums och namespaces som innehåller värden får Node.js att avbryta med `ERR_UNSUPPORTED_TYPESCRIPT_SYNTAX`, och en typ som importeras utan nyckelordet `type` behandlas som en import av ett värde, som fallerar vid körning. Slå på `erasableSyntaxOnly` och `verbatimModuleSyntax` så att kompilatorn rapporterar båda innan Node.js gör det, skriv `import type` för typer och ersätt enums med objektliteraler markerade `as const`.

Läs mer

- TypeScript Documentation — handboken, referenssidorna och versionsanteckningarna, underhållna av TypeScript-teamet.
- The Basics — handbokens kapitel om statisk kontroll, utdata trots fel, raderade typer och strikthet.
- TSConfig Reference: strict — varje kompilatoralternativ med sitt standardvärde, bl.a. de kontroller som flaggan strict slår på.
- Type Compatibility — strukturell subtypning och de ställen där typsystemet med avsikt är osunt.
- Node.js: Modules: TypeScript — hur typerna skalas bort, den syntax som avvisas och de kompilatorinställningar som Node.js rekommenderar.
- Linting with Type Information — hur typescript-eslint ger lintregler tillgång till typkontrollen.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är TypeScript?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/typescript/