Engineer

TypeScript

実行前に型を検査するJavaScriptの拡張言語

どのようなものか

TypeScriptは、JavaScriptに型注釈を加え、それを検査するコンパイラー `tsc` を組み合わせた、Microsoftのプログラミング言語です。TSとも略され、ファイルの拡張子は `.ts` 、JSXのマークアップを含む場合は `.tsx` です。JavaScriptの構文はTypeScriptでも有効なので、JavaScriptのスーパーセットと呼ばれます。コンパイラーはApache License 2.0の下で公開されているオープンソースです。

型が存在するのは、検査している間だけです。通常の型注釈は、TypeScriptがJavaScriptを出力するときに消去されます。実行時の検証を提供するものではなく、実行時に型を検査するライブラリも必要としません。型注釈を加えても、実行中のJavaScriptの動作は変わりません。この一点が、価値と限界の両方を説明します。検査器が推論するのは、ソースコードに記述された型についてであり、実行中に届くデータを検証するわけではありません。

型システムは構造的です。名前が何であっても、構造が合えば二つの型は互換と見なされます。これは、単純なオブジェクトを受け渡すJavaScriptの使い方に適しています。対して、C#やJavaは宣言された名前を比較します。型の多くは推論されるため、注釈は主に関数のシグネチャや公開する値に付きます。また、型システムは一部で意図的に健全性を保証していません。 `any` はその値の検査を無効にし、 `as` によるアサーションは、その型についてプログラマーの申告を信じるようコンパイラーに指示します。どちらも、ほかの型情報とともに出力から消えます。

実行方法は大きく三つあります。 `tsc` はプロジェクトを検査してJavaScriptを出力します。esbuildなどのバンドラーは、型を検査せずに取り除き、検査は別に実行する `tsc --noEmit` に任せます。Node.jsはv23.6.0とv22.18.0以降、型の構文を空白で置き換えて `.ts` ファイルを直接実行します。この場合も型検査はせ

ず、その方法で消去できる構文だけを受け入れます。2026年7月に発表されたTypeScript 7.0は、Goで書かれたネイティブ版コンパイラーです。開発チームは、完全なビルドでは通常8〜12倍速いと報告しています。

概念を判断に生かす

「TypeScriptとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 読み込みが速く、スマートフォンでも使え、お客様がどのブラウザーを使っていても同じように動く画面を提供します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 フルスタックのGIS地図開発を指揮 — DBからブラウザーまでのフルスタック地図開発

フルスタックのGIS地図開発を指揮 お問い合わせ →

なぜ重要なのか

チームが判断するのは、実行前に誤りを見つけるために、ビルド工程と検査器を追加するだけの価値があるかです。複数人が毎週変更するコードベースでは、たいてい価値があります。フィールド名を変えると、それを使っていた箇所がすべて分かり、エディターは型を基にコードを補完し説明します。モジュールのシグネチャは、呼び出し側が気付かれずに破ることのできない契約になります。代償は、ビルドごとの検査、多数の選択肢がある設定ファイル、そして対象ライブラリの変更に追従し続ける型定義です。

最も重要な設定は、検査の厳格さです。`strict` フラグは、`strictNullChecks` や `noImplicitAny` など、一群の検査を有効にします。ハンドブックは、その結果をスイッチではなく、段階を調整するダイヤルとして説明しています。TypeScript 6.0以降、このフラグは既定で有効なので、新規プロジェクトは厳格な状態から始まります。古いプロジェクトではfalseのままのこともあり、後から有効にするには、影響するファイルを一つずつ確認し、対応する必要があります。便利な二つの検査、`noUncheckedIndexedAccess` と `exactOptionalPropertyTypes` は、この一群には含まれず、別に有効にします。

型の消去が、保護の限界を決めます。ネットワーク要求、ファイル、キュー、フォームからJSONを読むプログラムでは、そのデータの構造について頼れるのはプログラマーの申告だけです。データが届く時点で、型は消えているからです。そのため、境界では実行時の検証が必要になります。検査と型が食い違わないよう、理想的には型と同じ定義から生成します。TypeScriptはプログラム内部の整合性を保ち、境界の扱いはコードに委ねます。

検査器は、ほかのツールにも情報を渡します。typescript-eslintなどのリンターは、コンパイラーに型情報を問い合わせることで、誰も結果を待たないPromiseや、モジュール内に広がる `any` 型の値を検出できます。チーム独自の約束を、レビューの指摘ではなくエラーにする仕組みです。

実際の活用例

このサイトで紹介するプロジェクトでは、TypeScriptの信頼性を、慣習だけに頼らず、生成コードとツールで支えています。三つのページが、その方法を示します。

あるプラットフォームでは、フロントエンドのTypeScriptクライアントを手書きしません。API自身の記述から生成します。その記述はHumaがGoの型から出力し、Goの型はデータベースのスキーマと関数に従います。記録された実装段階で、クライアントは44,076行あり、同時に生成した61個のモックハンドラーにより、フロントエンドのテストは手書きのフィクスチャではなく、実際の契約に対して動きました。各段階に独自の検査があります。CIの契約検査が、フロントエンドの送信内容とAPIが期待する内容を比較し、食い違えばビルドを失敗させます。その下では、スキーマの検査処理が実際の構造を調べます。欠落とnullを混同しやすい省略可能なリクエスト本文のフィールドや、クエリーパラメーターの列挙値も対象です。片側でフィールドが変わると、不一致を出荷した後でブラウザーに422が現れるのではなく、変更から数分でビルドが失敗します。代わりに、契約の変更をクライアントの型へ反映するには、連鎖をもう一度実行する必要があり、再生成には手間がかかります。

別のページは、TypeScriptを含む六つの言語に、同じ基準を設けています。ESLintの診断はすべてエラーで、警告扱いにして逃がす段階はありません。コード内での抑制も禁止するため、指摘は黙らせるのではなく直す必要があります。コミット前とプッシュ前のフックがリンターとテストを実行し、関数とファイルの長さの上限が、モジュールの読みやすさを保ちます。

言語に付属するリンターだけでは、プロジェクト固有の規則まで扱えません。そこで、記録された実装段階では、十五個の独自ESLintルールが、そのリポジトリのTypeScriptの書き方を守っていました。これは274本の検査スクリプトからなる仕組みの一部で、うち268本はコミットフックに登録されていました。検査自体にも146件のテストがありました。検査が壊れると、誰にも気付かれずにすべてを通してしまうからです。型ではなくテストを扱うルールもあります。 `fix:` で始まるコミットには、その修正がなければ失敗するテストが必要です。レビュー時間は機械的な確認から設計へ移り、代わりにコミットは遅くなりました。

問題が起きるところ

TypeScriptの失敗の多くは、コンパイラーから見える範囲を超えて型を信じることから生じます。

文字列型のフィールドが、実行時にはundefinedだった。 データはプログラムの外から届き、検査する代わりに型を与えられています。DOMライブラリは `res.json()` を `any` のPromiseとして型付けするため、結果をどう使ってもコンパイラーは受け入れます。アサーションは、その信頼を明示するだけで、安全にはしません。

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

外部データは `unknown` として扱い、境界で実行時にスキーマ検証を行い、そのスキーマから型を生成して両者をそろえます。型情報を使うリンタールールなら、すり抜けた `any` も検出できます。

型エラーがあるコードが、そのまま本番へ届く。 `tsc` はエラーを報告してもJavaScriptを書き出します。 `noEmitOnError` の既定値がfalseだからです。バンドラーとNode.jsは、型をまったく検査せずに消去します。CIとコミット前フックで `tsc --noEmit` を必須工程にし、その終了コードに従ってビルドを失敗させます。

インデックスによる参照で、型が除外していたundefinedが返る。 配列要素やインデックスシグネチャを通じたキーの読み取りは、配列の末尾を越えていても、常に値があるものとして型付けされます。 `noUncheckedIndexedAccess` を有効にすると、その読み取りに `undefined` が加わり、コードでの対処が必要になります。 `exactOptionalPropertyTypes` と組み合わせると、プロパティが欠落している場合と、値が `undefined` である場合を区別できます。

tscでコンパイルできたファイルが、Node.jsでは失敗する。 型の除去で消えるのは、実行時の意味を持たない構文だけです。値を持つ列挙型や名前空間は、Node.jsを `ERR_UNSUPPORTED_TYPESCRIPT_SYNTAX` で停止させます。 `type` キーワードなしにインポートした型は、値のインポートとして扱われ、実行時に失敗します。 `erasableSyntaxOnly` と `verbatimModuleSyntax` を有効にして、Node.jsで動かす前にコンパイラーで両方を検出し、型には `import type` を使います。列挙型は、 `as const` を付けたオブジェクトリテラルに置き換えます。

参考資料

- TypeScriptドキュメント — TypeScriptチームが保守するハンドブック、リファレンス、リリースノートです。
- 基本事項 — 静的検査、エラーがある場合の出力、型の消去、検査の厳格さを扱うハンドブックの章です。
- TSConfigリファレンス：strict — コンパイラーの各オプションと既定値、strictフラグが有効にする検査を説明しています。
- 型の互換性 — 構造的サブタイピングと、型システムが意図的に健全性を保証しない箇所を扱います。
- Node.js：モジュール：TypeScript — 型の除去、拒否する構文、Node.jsが推奨するコンパイラー設定です。
- 型情報を使うリント — typescript-eslintが、リンタールールに型検査器へのアクセスを提供する仕組みを説明しています。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「TypeScriptとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/typescript/