目次

TypeScript コンパイラの基礎 9

第3章 ES2015+の基本構文

27

第4章 型演算の基本

67

第5章 高度な型演算

105

第6章 モジュールシステム

127

この章では、TypeScriptコンパイラおよびその主要な実装であるtscコマンドについて、基本的な使い方を解説します。実際のWebフロントエンドやバックエンドの開発ではtscを直接呼び出すことはまれですが、それでもtscはTypeScriptで開発するときの基本です。tscがどういうソフトウェアなのかは、知っておいたほうがよいでしょう。

ところで「TypeScriptコンパイラ」と「tsc」は必ずしも同じ意味ではありません。現在のところTypeScriptコンパイラはいくつか異なる実装があります。

tscはTypeScriptチームによる代表的な実装です。参照実装としての側面もあるため、TypeScriptの仕様はtscが実装しているものと考えてよいでしょう。しかしそれ以外にも、BabelのTypeScriptプラグインやesbuildもTypeScriptをコンパイルできます。これらはtscのすべての機能を実装しているわけではありません。tscと併用することが前提とされていますが、それでも十分に実用的です。

ただし実際には、TypeScriptコンパイラはtscとほぼ同じ意味で使われることも多いです。本書ではtscそのものについて触れるとき以外は「TypeScriptコンパイラ」で用語を統一しています。

2-1 nodeコマンドをインストールする

TypeScriptコンパイラの準備をする前に、コマンドラインのJavaScript処理系であるNode.jsが必要です。Node.jsのインストール方法はいくつかあります。

まず、公式サイト[注1]からビルド済みパッケージをダウンロードして手動でインストールするパターンです。この方法はOSによらず簡単にそのときの最新バージョンをインストールできます。しかし、この方法で継続的に最新バージョンをインストールするのは手間がかかります。

次に、OSごとの汎用的なパッケージマネージャで入れる方法があります。macOSであればHomebrew、Windows（WSL）であればaptやLinuxbrewなどです。HomebrewやLinuxbrewであれば最新版を入れられますし、その後継続して新しいバージョンに追従するのも簡単です。aptで入るNode（nodejsパッ

注1）https://nodejs.org/en/download/

ケージ）のバージョンはディストリビューションにもよりますが、少し古いことが多いでしょう。

また、nvmやnodebrewのようなNode.js専用のバージョンマネージャを使うこともできます。このような専用のバージョンマネージャを使うと、複数バージョンのNode.jsをインストールしてプロジェクトごとに使い分けることができます。

ところでNode.jsのメジャーバージョンは、開発版である「Current」バージョンと安定版である「LTS（Long Term Support：長期サポート）」バージョンがあります。Currentはその時点での開発版でいずれ安定版になる予定のバージョンですが、処理系自体が不安定なことがあります。実際の運用ではLTSを使い、Currentはテストで使うにとどめるのがよいでしょう。ただし、macOSのパッケージマネージャであるHomebrewではデフォルトでCurrentの最新版が入るようです。本書で扱う範囲においてはCurrentでも問題ありません。

またいずれの場合もNode.js用のパッケージマネージャであるnpmコマンドも同時にインストールされます。npmはもともとNode.js用プロジェクトのパッケージマネージャですが、現在はWebフロントエンド用のライブラリも管理できます。

COLUMN

Node.jsとブラウザ以外のJavaScriptランタイム

Node.js以外のJavaScriptランタイムもここで少し紹介します。「JavaScriptランタイム」というとき、純粋なJavaScript処理系に加えて何らかの実行環境やライブラリが付属しています。JavaScriptランタイムは大きく分けると次の3種類あります。

- **ブラウザ用**
- **コマンドラインアプリケーション用**
- **エッジコンピューティング用**

ブラウザ用のランタイムは、ブラウザに搭載されており、Webサイトを操作するためのランタイムです。もっとも一般的なJavaScriptランタイムともいえます。典型的には、純粋なJavaScript処理系に加えて、Web API（ブラウザAPI）を備えています。ブラウザ用のランタイムを想定したアプリケーションを「Webフロントエンド」と呼ぶこともあります。「Webフロントエンド用のライブラリ」は、ブ

本章では、ES2015+の基本構文を、TypeScriptで使うという文脈で解説します。ESとはECMAScriptの略で、ECMAScriptはJavaScriptの標準規格であるECMA-262で定義される言語の正式名称です。本書では、ECMA-262第6版[注1)]で定義される「ECMAScript 2015」を「ES2015」、そしてES2015を含むそれ以降の版のESを「ES2015+」と総称しています。本書執筆時点では最新のESはES2024なので、「ES2015+」とは、ここではES2015からES2024までのESすべてという意味です。

なお「ES2015+」としてES2015を特別扱いするのは、ES2015は長いESの歴史の中で特別なバージョンだからです。その前のメジャーバージョンであるECMA-262第5版（ES5）は2009年に策定されたのですが、ES2015はそれから紆余曲折を経て6年越しのメジャーアップデートでした。そしてES2015でクラスやESモジュール、テンプレート文字列など、言語仕様へ追加された新機能が非常に多かったのです。そのあと、ESは毎年バージョンアップされる体制に変化し、ES2015以降も多くの機能が追加されていますが、依然として「新しいJavaScript」として「ES2015+」という総称が使われています。

TypeScriptはES2015+との互換性を維持したまま発展してきました。したがって、TypeScriptを使うためには、ES2015+を学ぶ必要があります。本章はES2015以降のESの新機能、つまりES2015+について、TypeScriptで使うという文脈のもと解説します。

3-1 変数宣言

それでは、ES2015+で導入された新機能を、TypeScriptとの関係を絡めて解説していきます。まず、コードの全域に影響を与える変数宣言です。なお、本章はJavaScriptの機能の説明ですが、サンプルコードはとくに断りのない限りTypeScriptで書いています。

varによる変数宣言の復習

ES2015より前のJavaScriptでは、次のようにvarが唯一の変数宣言でした。

注1) http://www.ecma-international.org/ecma-262/6.0/

var.mts

```
var x = "Hello, world!";
console.log(x); // => Hello, world!
```

varは関数の中のどこで宣言しようとも有効範囲が関数全体となり、また変数を複数回宣言してもただ一度だけ宣言したのと変わりません。そしてこの仕様は多くの誤解とバグを生み出すことになりました。

たとえば、次のようなコードは文法的には有効です。この関数f()では複数回var xを宣言していますが、2度目に宣言しても何も効果はありません。その結果、xには"second"が代入されます。変数宣言をしたのに何も効果がないというのは、わかりにくい仕様だと思います。

var-file-scope.mts

```
function f() {
  var x = "first";
  if (true) {
    var x = "second";
  }
  console.log(x); // => second
}
f();
```

現代においてvarを使わなければならない状況はありませんが、varを使っている古いコードはまだあるため、プロジェクトによっては読む必要はあるかもしれません。しかし、本書でvarが出てくるのはこの節だけです。

ブロックスコープの変数宣言let

一方ES2015+では、コードブロックごとに新しい変数定義が作られるletとconstを使います。コードブロックごとに定義が作られるため、ブロックスコープの変数宣言と呼ばれます。letは再代入可能で、constは再代入不可能な変数宣言です。

まず、先ほどのvarを使ったコードをletに置き換えてみると、"first"が印字されます。constでも結果は同じです。

let.mts

```
function f() {
  let x = "first";
  if (true) {
```