環境構築

目次

※サンプルコードはこちら

1. Node.js をインストール

- Node.js ダウンロードサイト
- バージョン 14.16.2 LTS

> `node -v`コマンドでインストールした node のバージョンを確認できる。

2. TypeScript をインストール

```
npm install -g typescript ts-node @types/node
```

> TypeScript 本体は、npm(Node Package Manager)からインストール可能。コマンドプロンプトから上記のコマンドを実行。`-g`オプションは、ライブラリをグローバルにインストールするために指定。

npm	役割
`ts-node`	指定された TypeScript のファイルをコンパイルして実行するための CLI ツール
`typescript`	TypeScript のコンパイラー。
`@types/node`	Node.js 由来の機能やモジュールに対する型情報

参考情報

- npm 上の ts-node 情報
- npm 上の typescript 情報
- npm 上の@types/node 情報

3. フォルダとファイルを作成し、実行

- `src`フォルダを用意し、`src/index.ts`のファイルに`console.log("Hello World");`と書く
- コマンドプロンプトに `npx ts-node src/index.ts` と書く
- コマンドプロンプトに`Hello World!`と出力されることを確認

> npx は勝手にインストールされているパスを調べて「コマンド名」が存在するかチェックして実行してくれる。

4. プロジェクトディレクトリの作成

- mkdir コマンドでディレクトリを作る
  - `mkdir typescript-node`
- cd コマンドで指定のディレクトリ配下のカレントディレクトリを移動する
  - `cd typescript-node`
- package.json の生成
  - `npm init -y`

> npm を利用するために、プロジェクトで利用する npm モジュールを`package.json`で管理する必要がある。このファイルは、プロジェクトの名前や利用する npm モジュールの依存関係が記載されている。`npm init`コマンドを使うと、対話形式でファイル名やライセンス情報などを作成できる。`-y`は`-yes`の省略版で全て「yes」を選択した状態で作成する。

- パッケージのインストール
  - `npm install typescript`
  - `npm install -D @types/node`

> `-D`は`--save-dev`の略でオプション。package.json の`devDependencies`にパッケージ名が記載される

- tsconfig.json の生成
  - `npx tsc --init`
- `tsconfig.json`ファイルを確認 以下のようになるようにコメントを修正

```
{
  "compilerOptions": {
    "target": "es6",
    "module": "commonjs",
    "strict": true,
    "esModuleInterop": true
  },
  "include": ["src/**/*"]
}
```

設定項目名	設定値	意味
target	es6	コンパイル後の JavaScript のバージョン
module	commonjs	どの方式でモジュール関連のコードを扱うか
strict	true	TypeScript の基本的なチェックを全て true にするか
esModuleInterop	true	import 文を使って読み込めるようにするか
include	["src/**/*"]	src 配下にあるファイルを一括で指定

5. TypeScript ファイルを作成し、生成された index.js を実行

- `typescript-node`配下に`src` フォルダを作成
  - `mkdir src`
- src フォルダへカレントディレクトリを移動
  - `cd src`
- `index.ts`ファイルを作成
  - `type nul > index.ts`
- `index.ts`に以下のコードを書く

```
function greeting(name: string): string {
  return `Hello, ${name}!`;
}
console.log(greeting("Typescript"));
```

- tsc コマンドでコンパイルをする
  - `npx tsc index.ts`
- `index.js`ファイルが生成されたことを確認
- node で`index.js`ファイルを実行し、結果を確認
  - `node index.js`

6. ts-node で手間を減らす

毎回 ts ファイルを編集して tsc コマンドでコンパイルし、node で実行するのは、手間もかかる上に面倒なので tsc→node の実行を自動で行ってくれる`ts-node`を利用する

- コマンドを実行してみる
  - `npx ts-node index.ts`

7. `ts-node-dev`でさらに開発効率を上げる

必要なファイルのいずれかが変更されると(標準として)ターゲットノードプロセスを再起動するが、再起動間で TypeScript コンパイルプロセスを`node-dev`で共有する。 毎回コンパイルをインスタンス化する必要がないため、再起動速度が向上する

- `ts-node-dev`をインストールする
  - `npm install -g ts-node-dev`
- 実行する
  - `npx ts-node-dev --respawn index.ts`

> サーバーのように一度実行すると動き続けるものではなく、一度実行するとプロセスが終了するプログラムの場合は、`ts-node-dev`に`--respawn`オプションをつける

- `src/index.ts`を修正して保存
- 自動で再コンパイル&実行
- `ts-node-dev`を終了するときは`Ctrl+C`を押下

参考情報

- npm 上の ts-node-dev 情報
- ts-node-dev の github

補足説明

Node.js とは

Google Chrome と同じV8という実行エンジンで作られた JavaScript の実行環境。 本来は、 JavaScript はブラウザでしか動作しないが、Node.js を使うことで、JavaScript を様々な場所で実行できる。

- テキストファイルの node プログラムを動かすコマンド
  - node ファイル名

偶数バージョンと奇数バージョンの違い

Node.js には 2 種類のバージョンがある

バージョン種類	**概要**
偶数バージョン	LTS(Logn Term Support の略)。長期サポートが保証されている
奇数バージョン	新しい技術を積極的に取り入れるバージョン。バグがあったり、1 年未満でサポート終了の可能性あり。

npm とは

Node Package Manager の略。Node には様々な拡張機能がパッケージという形で配布されている。これらのパッケージを一つの場所に集約させ、開発者が必要なものだけ自由にインストールできるような仕組み。パッケージ管理ツール。また、開発者のローカルにある package.json を見ることで、プロジェクトで何のパッケージを使っているかも分かる。

npx とは

Node Package Executer の略。パッケージの実行を行うツール。`npx` は特定のパッケージをインストールせずに使用するもの。今回は特定のプロジェクトの中に`ts-node`をインストールせずに使用しているので`npx`での実行を行っている。

package.json の devDependencies とは

開発時に必要なパッケージが記載される。つまり、公開時には不要なパッケージ。今回インストールしたパッケージ`@types/node`は JavaScript のライブラリに型定義を付与する、型定義ファイルであり、開発時のみ使う情報なのでインストール時に`-D`とし、`devDependencies`に入れている。