



全2回で学ぶ TypeScript入門講座 1回目

2025/7/8

Kazuma SEKIGUCHI

自己紹介



関口和真

株式会社コムセントCTO

Webシステム開発、スマートフォンアプリ制作、
サーバー構築、運用など

スマートフォンを使ったアプリの制作

サーバーサイドシステムの作成

フロントエンド部分の作成

目標

- TypeScriptはどのようなものであるかを知る
- TypeScriptの型を知る
- ジェネリクスの使い方を知る
- 条件型と型ガードを知る
- 型定義ファイルとの付き合い方を知る



今回のアジェンダ

- TypeScriptとは？
- 型システムの理解
- 基本的な型
- 関数の型

TypeScriptとは

- JavaScriptに静的型付けなどの機能を追加したMicrosoftが作成したプログラミング言語
 - TypeScriptのコードはJavaScriptに変換（トランスパイル）して実行する
- インターフェイスやクラスへのアクセス修飾子なども追加されているため、大規模な開発に向いている
- 一般的に代替JS言語（AltJS言語）の1つとされる
 - CoffeeScript
 - Dartなども存在する

TypeScriptとは

- 現在のAltJS言語の代表格
 - 現在はJSでそのまま記述することが少なくなりつつあり、TypeScriptで記述するケースが増えてきている
 - 複雑なJSコードが増えるほどTypeScriptを利用した方が効果が高いとされる
- TypeScriptはJavaScriptをそのまま記述することも可能
 - 一部分だけTypeScriptの記法を利用することもできる
 - 学習しつつJSからTypeScriptへの移行が可能になるため、移行コストが比較的少ない

そもそも型とは

- 型とは変数などにどのようなデータ形式を格納しているかを定義しているもの
 - 例えば、文字列型、数値型、真偽値型などが存在する
 - 変数の型が異なることで挙動が異なってくる

```
const num1 = '20';//文字を格納している
const num2 = 30;//数字を格納している
const num3 = 40; //数字を格納している
const total1 = num1 + num2;//2030となる
const total2 = num2 + num3;//70となる
```


JSの型

- JSは動的型付け言語
 - 変数などに対してどういうデータを格納したかによって、型が自動的に決まる
 - 文字列を格納すれば、文字列型になる
 - 型は変更することが可能（ものによっては不可能）
 - 文字列を数字にするのは状況によってはおかしい結果となる
- 変数に格納した初期値によって型が決まるが、後から想定外の値を入れたときに型が変わる
 - `const`は上書きできないので、変わらない

JSの型

- 型が変わることがある場合、プログラマーによって上書きされた場合、気付かない可能性がある
 - 最初は文字列を格納して、後から数字で上書きした
 - 他のプログラマーが最初の格納値だけを見たら文字列型と解釈するが実際には数字型となり、バグが起きやすくなる
- 初期化するときに変数に対し型を紐付けるようにすることで、上書きすることを避けることが可能
 - TypeScriptは変数や引数に型を手動で付けるため、静的型付け言語という扱いになる
 - 文字列を格納したら後から数字を上書きした時点でエラーになる
 - TypeScript対応のエディターであれば、上書きした時点でエラーを検知することが可能

関数の引数

- JSでは関数に引数を指定できるが型は指定できないため、
 というデータが入ってくるかは分からない
 - コメントなどで型を記述しておくことは可能だが、プログラムとしては自動的に判断できない
 - TypeScriptでは引数に型を指定できるため、型が違う値が入ってきた場合にはその段階でエラーとなる
 - 想定外のデータが入ってくることができる
- 関数の戻り値の型も指定することが可能
 - 戻り値に型を指定できることで、戻り値を受け取る変数で型が異なっていたらエラーにできる

エラーの是非

- 型が異なるとエラーになる
 - エラーが多くなるため、嫌になることもあるが、実行時にバグが生じるよりもエラーをきちんと修正することでバグの発生を減らすことができる
 - 基本的に開発時点でエラーが出てくるため、修正もしやすい
- コードヒントが正確になる
 - 最近のエディターであれば、記述する関数や変数などを1文字入力するとヒントとして提示してくれる
 - 静的型付け言語を使うと、より正確なものが提唱されてくる
 - 引数に指定する値の型も提示してくれるものが多い

型アノテーションの例

- TypeScriptでは変数や関数の引数、返り値に型を明示的に指定可能
 - 型アノテーションと呼ぶ

```
let num1:number = 100;  
num1 = 'abc';//→型が異なるためエラーとなる。  
let str1:string = 'abc';  
str1 = 'xyz';//上書きされる(型が同一)
```

型エイリアスの例

```
type User = {  
  id:number;  
  name:string;  
  email:string;  
  active:boolean;  
};
```

```
function showUserData(user:User):void{  
  if(user.active){  
    console.log(`${user.name}(${user.email})`);  
  }  
}
```

- typeを利用して独自の型を作成することが可能
 - 引数などでも指定可能
- functionの返り値がない場合は、voidを指定する

よく利用する型

型名	種類
string	文字列
number	数値(整数・小数)
boolean	真偽値(true,false)
any	全ての型(できるだけ避けるべき)
undefined	未定義
null	値が存在しない
void	関数の返り値で返り値がない場合

配列の型、ユニオン型

- 配列の中身の型も指定することが可能

```
let num:number[] = [1,2,3];  
let name:Array<string> = ['Tom','Bob','Brown'];
```

- 変数が複数の型を取るように指定することも可能
 - ユニオン型

```
let data:string | number;  
data = 'JavaScript';//OK  
data = 24;//OK  
data = true;//NG
```

パイプを使って型を複数
指定することが可能

booleanは型として許され
ていないためエラーになる

タプル

- 要素と型の順序が固定された配列
 - 関数の戻り値として使われることが多い

```
function getUserInfo():[string,number]{  
  return ['Tom',35];  
}
```

戻り値としてstring,numberの順番
で値が返ってくる

```
const [name,age] = getUserInfo();
```

```
const nums = [string,...number[]] = ['点数',67,87,54];
```

- タプルの末尾に...を記述することで、2番目以降の型を指定することが可能

enum型 (列挙型)

- TypeScriptでは列挙型が存在する
 - 変数で格納できる値を事前に指定しておくことができる
 - 数値列挙型は0から順に数字が割り当てられる

```
enum Direction{  
  Up,  
  Down,  
  Left,  
  Right  
}
```

数値列挙型

```
function move(dir:Direction):void{  
  console.log("MoveTo:"+Direction[dir]);  
}  
move(Direction.Up);//MoveTo:Upと表示される
```

enum型 (列挙型)

- 文字列列挙型の場合、デバッグ時に値がわかりやすい
 - 数値列挙型の場合、0, 1と表示される
 - 文字列列挙型であれば、"UP", "DOWN"と表示される

```
enum Direction {  
  Up = "UP",  
  Down = "DOWN",  
  Left = "LEFT",  
  Right = "RIGHT",  
}
```

文字列列挙型

```
function move(dir:Direction):void{  
  console.log("MoveTo:"+dir);  
}  
move(Direction.Up);//MoveTo:UPと表示される
```

型推論

- TypeScriptでは明示的に型を指定することもできるが、指定しなくても問題無い
 - 型推論が動作する
 - 変数にどういう値を入れるかによって、型推論が働き、型が指定される
 - 型は指定されるため、型が異なるものを後から入れることはできない
 - ただし、anyと型付けされることもあるので注意が必要
- 個人的には明示的に記述したほうが無難だと考える
 - 明示的に記述しておくことで、判別が容易になる

関数型

- JSの場合、関数を変数に代入して利用することが可能
 - 関数型を使うことで型を指定することが可能

```
const multiply = (a: number, b: number): number => {  
  return a * b;  
};
```

- typeの型エイリアスを利用して型を指定することもできる

```
type Multiply = (a: number, b: number) => number;  
const multiply: Multiply = (x, y) => {  
  return x * y;  
}
```

関数

```
function greet(name?: string): void {  
  console.log(`Hello, ${name ?? "Guest"}`);  
}
```

- ?を付けることで引数の省略が可能
 - 引数を与えなくても関数を動作させることができる

```
function greet(name: string = "Guest"): void {  
  console.log(`Hello, ${name}`);  
}
```

- 引数にデフォルト値を指定することが可能
 - 引数に値を与えることで上書きが可能

TypeScriptの作成

- NPMでTypeScriptを導入する
 - グローバルにインストールする方法と、ローカルにインストールする方法が存在
 - 今回はグローバルにインストールする

```
npm install -g typescript
```

TypeScriptをグローバルにインストールする

```
tsc -v
```

TypeScriptのバージョンを確認する

```
npx tsc --init
```

作業フォルダー上で実行することで、tsconfig.jsonファイルが生成される

TypeScriptでのプロジェクト作成

- フォルダーを作成
- ターミナル等でフォルダーに移動し、

```
npm init -y
```

でpackage.jsonを作成

- TypeScriptの設定ファイルを作成する

```
npx tsc --init
```

作業フォルダー上で実行することで、
tsconfig.jsonファイルが生成される

- tsconfig.jsonというTypeScriptの動作条件を設定するためのファイルを再生する

TypeScriptからJSへの変換

- TypeScriptは拡張子を.tsとして保存する
 - トランスパイルすると、.jsファイルとして生成される

`npx tsc index.ts`

index.tsをindex.jsにトランスパイルする

- index.jsが生成されるので、必要に応じてhtmlに読み込ませたりして実行する

tsconfig.jsonの設定

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "ES2022",
    "moduleResolution": "Node",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "noUncheckedIndexedAccess": true,
    "resolveJsonModule": true,
    "isolatedModules": true,
    "verbatimModuleSyntax": true,
    "incremental": true,
    "outDir": "dist",
    "rootDir": "src",
    "sourceMap": true
  },
  "include": ["src/**/*.ts"],
  "exclude": ["node_modules", "dist"]
}
```

- 現状ブラウザーで実行するJSを生成するなら左側の設定で良いはず
- targetで出力するJSのバージョンを指定することが可能
 - ターゲットとするブラウザーに合わせて変更する
 - es2016,es2020などが指定可能
- moduleも同様
 - ES2022,esnextなどを指定
- Nodeでも実行するなら、moduleをNodeNextにする

tsconfig.json

- tsconfig.jsonを作成しておくと、コマンドでtsファイル全体をトランスパイルしてくれる

`npx tsc`

tsファイルを全てjsにトランスパイルする

変数名

- たまにTypeScriptで記述すると、変数名が使えないことがある
 - TypeScriptで予約語として扱われてしまっている
 - 「ブロックスコープの変数　〇〇を再宣言できません」とVSCodeにはエラーが出てくる
 - 一番簡単なのは、別の変数名に変えること
- lib.dom.d.tsを使わないようにする、という手もあるが、ブラウザーで動作しなくなるためオススメしない

ありがとうございました。
また次回。