変数

TypeScriptとJavaScriptの一番の違いは型です。型が登場する場面は主に3つです。

- 変数(プロパティも含む)
- 関数の引数
- 関数の返り値

本章ではまず変数について触れ、TypeScriptの型システムの一部を紹介します。

関数については関数の章で説明します。 型システムの他の詳細については、オブジェクトの型付け（インタフェースの章）、クラスの型付け（クラスの章）、既存パッケージへの型付けの各章で説明します。

三種類の宣言構文

変数宣言には `const` 、 `let` があります。 `const` をまず使うことを検討してください。 変数は全部とりあえず `const` で宣言し、再代入がどうしても必要なところだけ `let` にします。 変わる必要がないものは「もう変わらない」と宣言することで、状態の数が減ってコードの複雑さが減り、理解しやすくなります。 変数を変更する場合は `let` を使います。 なお、この `const` は、多くのC/C++経験者を悩ませたオブジェクトの不変性には関与しないため、再代入はできませんが `const` で宣言した変数に格納された配列に要素を追加したり、オブジェクトの属性変更はできます。 そのため、使える場所はかなり広いです。

変数の使い方

```
// 何はともあれconst
const name = "小動物";

// 変更がある変数はlet
// 三項演算子を使えばconstにもできる
let name;
if (mode === "slack") {
    name = "小型犬";
} else if (mode === "twitter") {
    name = "小動物";
}
```

なお、JavaScriptと異なり、未定義の変数に代入すると、エラーなります。

```
undefinedVar = 10;
// error TS2552: Cannot find name 'undefinedVar'.
```

若者であれば記憶力は強いので良いですが、歳をとるとだんだん弱ってくるのです。また、若くても二日酔いの時もあるでしょうし、風邪ひいたり疲れている時もあると思うので、頑張らないで理解できるように常にしておくのは意味があります。

注釈

昔のJavaScriptは変数宣言で使えるのは `var` のみでした。 `var` はスコープが関数の単位とやや広く、影響範囲が必要以上に広くなります。 また、宣言文の前にアクセスしてもエラーにならなかったりと、他の宣言よりも安全性が劣ります。 現在でも使えますが、積極的に使うことはないでしょう。

変数の使い方

```
// 古い書き方
var name = "小動物";
```

変数の型定義

TypeScriptは変数に型があります。TypeScriptは変数名の後ろに後置で型を書きます。これはGo、Rust、Python3などでみられます。 一度定義すると、別の型のデータを入れると、コンパイラがエラーを出します。それによってプログラムのミスが簡単に見つかります。 また、型が固定されると、Visual Studio Codeなどのエディタでコード補完機能が完璧に利用できます。

変数への型の定義

```
// nameは文字列型
let name: string;

// 違う型のデータを入れるとエラー
// error TS2322: Type '123' is not assignable to type 'string'
name = 123;
```

なお、代入の場合には右辺のデータ型が自動で設定されます。これは型推論と呼ばれる機能で、これのおかげで、メソッドの引数や、クラスや構造体のフィールド以外の多くの場所で型を省略できます。

推論

```
// 代入時に代入元のデータから型が類推できる場合は自動設定される
// 右辺から文字列とわかるので文字列型
let title = "小説家";

// 代入もせず、型定義もないと、なんでも入る（推論ができない）any型になります。
let address;
// 明示的に any を指定することもできる
let address: any;
```

型については型の章で取り上げます。 変数以外にも関数の引数でも同様に型を定義できますが、これについては関数の節で紹介します。

より柔軟な型定義

TypeScriptは、JavaやC++、Goなどの型付き言語を使ったことがある人からすると、少し違和感を感じるかもしれない柔軟な型システムを持っています。 これは、型システムが単にプログラミングのサポートの機能しかなく、静的なメモリ配置まで面倒を見るような言語では不正となるようなコードを書いても問題がないからと言えるでしょう。 2つほど柔軟な機能を紹介します。

- AでもBでも良い、という柔軟な型が定義できる
- 値も型システムで扱える

AでもBでも良い、というのは例えば数値と文字列の両方を受け取れる（が、他のデータは拒否する）という指定です。

数字でも文字列でも受け取れる変数

```
// 生まれの年は数字か文字列
let birthYear: number | string;

// 正常
birthYear = 1980;
// これも正常
birthYear = '昭和';
// 答えたくないのでnull・・・はエラー
birthYear = null;
// error TS2322: Type 'null' is not assignable to type 'string | number'.
```

次のコードは、変数に入れられる値を特定の文字列に限定する機能です。 型は `|` で複数並べることができる機能を使って、取りうる値を列挙しています。 この複数の状態を取る型を合併型（Union Type）と呼びます。 ここで書いていない文字列を代入しようとするとエラーになります。 数値にも使えます。

変数に特定の文字列しか設定できないようにする

```
let favoriteFood: "北極" | "冷やし味噌";
favoriteFood = "味噌タンメン"
// error TS2322: Type '"味噌タンメン"' is not assignable to
//   type '"北極" | "冷やし味噌"'.

// 数値も設定可能
type PointRate = 8 | 10 | 20;
// これもエラーに
let point: PointRate = 12;
```

型と値を組み合わせることもできます。

```
// 値と型の合併型
let birthYear: number | "昭和" | "平成";
birthYear = "昭和";
```

変数の巻き上げ

`var`、`const`、`let` では変数の巻き上げの挙動が多少異なります。`var` はスコープ内で宣言文の前では、変数はあるが初期化はされてない（`undefined`）になりますが、他の2つはコンパイルエラーになります。宣言前に触るのは行儀が良いとは言えないため、`const`、`let`の挙動の方が適切でしょう。

変数の巻き上げ（変数の存在するスコープの宣言行前の挙動）

```
// 旧: varはundefinedになる
function oldFunction() {
  console.log(`巻き上げのテスト ${v}`);
  var v = "小公女";
  // undefinedが入っている変数がある扱いになり、エラーならず
}
oldFunction();

// 新: let/const
function letFunction() {
  console.log(`巻き上げのテスト ${v}`);
  let v = "小公女";
  // 宣言より前ではエラー
  // error TS2448: Block-scoped variable 'v' used before its declaration.
}
letFunction();
```

変数のスコープ

以前は`{`、`}`は制御構文のためのブロックでしかなく、`var`変数は宣言された`function`のどこからでもアクセスできました。`let`、`const`で宣言した変数のスコープは宣言されたブロック（`if`、`for`は条件式部分も含む）の中に限定されます。スコープが狭くなると、同時に把握すべき状態が減るため、コードが理解しやすくなります。

```
// 古いコード
for (var i = 0; i < 10; i++) {
  // do something
}
console.log(i); // -> 10

// 新しいコード
for (let i = 0; i < 10; i++) {
  // do something
}
console.log(i);
// error TS2304: Cannot find name 'i'.
```

なお、スコープはかならずしも制御構文である必要はなく、`{`、`}`だけを使うこともできます。

```
function code() {
  {
    //この変数はこの中でのみ有効
    const store = "小売店";
  }
}
```

まとめ

本章では、TypeScriptの入り口となる変数について紹介しました。昔のJavaScriptとはやや趣向が変わっているところもありますが、新しい`let`、`const`を使うことで、変数のスコープをせばめ、理解しやすいコードになります。