型定義

目次

0.構文：組み込み型

変数の宣言

```
let name: type = initial;
//name:変数の名前
//type:データ型
//initial:初期値
```

型の種類	概要
number	数値型(整数/浮動小数点数)
string	文字列型
boolean	真偽型(true/false)
symbol	シンボル型
any	任意の型(任意の型を許容)
オブジェクト型	後述

- TypeScript では、変数宣言で型が省略された場合にも、初期値から型を推論する。このことを型推論という。

1.構文：テンプレート文字列

文字列リテラルは以下のいずれかで括る

- シングルクウォート (')
- ダブルクウォート(")
- バッククウォート(`)

バッククウォートは文字列に変数を埋め込める、複数行にまたがる文字列を表現可能。

```
let mail: string = "dummy@dummy.com";

let msg = `お問い合わせ：質問がある方は気軽にご連絡ください。（メール送付先：${mail}）`;
```

1_am_samplecode_1.ts

2.構文：型アサーション

互換性のある型であれば<型名>という記述を変数/リテラルの前に付与することで、型を明示的に変更。型キャストという。

```
function output(result: string) {
  return `結果は${result}です。`;
}
// a. 数値型を渡すとエラー
output(20);

// b. any型に変換
output(<any>20);

// c. as構文でも置き換え可
output("20" as any);
```

1_am_samplecode_2-1.ts

以下の例は、コンパイルは通るが実行時にエラーになる。 実体は *string* 型なので *toRound* メソッドを呼び出せない

```
function toRound(value: number) {
  // b. 四捨五入する
  return Math.round(value);
}
// a. number型の引数にstring型の値を渡す
console.log(toRound(<any>"hoge"));
```

1_am_samplecode_2-2.ts

3.構文：変数のスコープ

変数の種類	概要
let	ブロックスコープに対応
var	ブロックスコープにを認識しない

let を使用するメリット

- 変数の範囲を最小限に納められる(=名前重複を最小限に抑えられる)
- 名前の重複が発生した場合もこれを検出できる

```
if (true) {
  var i: number = 1;
}
//ブロック外でも有効
console.log(i);

if (true) {
  let s: number = 2;
}
//ブロック外はコンパイルエラー
console.log(s);
```

1_am_samplecode_3.ts

変数の宣言には`var`より`let`を利用すべき

4.構文：定数宣言- const

一度格納した値をあとから変更できない入れ物=定数

定数の宣言

```
const name: type = initial;
//name:定数の名前
//type:データ型
//initial:初期値
```

再代入はできない

```
const ID: number = 123;
ID = 111;
//再代入不可
```

1_am_samplecode_4-1.ts

配列オブジェクト(IDs[i]の値)は定数ではないので、変更可能(オブジェクトでプロパティを変更する場合も変更可能) ただし、*IDs* 自体の変更は不可

```
const IDs = [123, 222, 345];
IDs[0] = 111;
console.log(IDs);

//結果：[111,222,345]

const IDss = [123, 222, 345];
IDss = [111, 222, 333];
console.log(Idss);

//  結果エラー
```

1_am_samplecode_4-2.ts

5.構文：配列

配列の宣言

```
let name: type[] = initial;
//name:配列の名前
//type:要素のデータ型
//initial:初期値(配列リテラル)
```

```
let language: string[] = ["JavaScript", "HTML", "CSS"];
console.log(language[0]);

//上記と同義
let language2: Array<string> = ["JavaScript", "HTML", "CSS"];
```

1_am_samplecode_5-1.ts

`readonly`キーワードを使用することで、読み取り専用の配列を定義することもできる。(*readonly* キーワードがチェックするのは一次元目の要素のみ)

```
let language: readonly string[] = ["JavaScript", "HTML", "CSS"];

//エラーになる
language[0] = "TypeScript";
```

1_am_samplecode_5-2.ts

6.構文：連想配列

文字列など意味あるキーで要素を管理する=連想配列(ハッシュ)

連想配列の宣言

```
let name: { [index: i_type]: v_type } = initial;
//name:配列の名前
//i_type:インデックスのデータ型
//v_type:値のデータ型
//initial:初期値(オブジェクトリテラル)
```

```
let lang: { [index: string]: string } = {
  eng: "英語",
  viet: "ベトナム語",
  por: "ポルトガル語",
};

//プロパティ構文
console.log(lang.por);

//ブラケット構文
console.log(lang["por"]);
lang.jpn = "日本語";
```

1_am_samplecode_6-1.ts

{[index:i_type]:v_type}インデックスシグニチャでの型宣言をする理由

- 型を明示的に指定しない場合、型推論で型を特定する
- eng/viet/porという文字列型のプロパティを持ったオブジェクト型と推論されたためエラー。

```
let lang = {
  eng: "英語",
  viet: "ベトナム語",
  por: "ポルトガル語",
};
//プロパティjpnは存在しないというエラーがでる
lang.jpn = "日本語";
```

1_am_samplecode_6-2.ts

7.構文：列挙型

関係する定数を束ねる

```
enum ename{name,...}
//enum:列挙型の名前
//name:定数
```

列挙子=値とすることで、列挙子に対して値を割り当てることも可能。 `height='高い'`のように数値だけでなく文字列の割り当ても可能

```
enum Priorities {
  HIGH,
  MIDDLE,
  LOW,
}

let p: Priorities = Priorities.HIGH;
console.log(p); //結果：0
console.log(Priorities[p]); //結果：HIGH

enum Priorities2 {
  HIGH = "高",
  MIDDLE = "中",
  LOW = "低",
}
let p2: Priorities2 = Priorities2.HIGH;
console.log(p2); //結果：高
console.log(Priorities2[p2]); //エラーとなる
```

1_am_samplecode_7.ts

8.構文：タプル型

- 複数の異なる型の集合を表現するためのデータ型。配列リテラルのような形式`[type1,type2,...]`=個々の要素の型を表す
- `readonly`キーワードを利用することで、読み取り専用のタプルを生成することも可能

```
let person: [string, number, boolean] = ["Miku", 68.92, true];

console.log(person[0].substring(2)); //結果：ku
console.log(person[1].toFixed(1)); //結果：68.9
console.log(person[2].toFixed(1)); //エラー
console.log(person[2] == false); //結果：true

//タプルの濫用を避けるべき理由
let person2: [string, number, boolean] = ["Miku", 68.92, true];
person2.shift(); //先頭の要素を削除
console.log(person2[0].substring(2)); //実行時エラー
```

1_am_samplecode_8.ts

タプルの濫用を避けるべき理由

配列 data の先頭を削除しても、データ型に反映されない。 `person2[0]`は依然と`string`型とみなされる。コンパイルは通るが実行時エラーとなる。