TypeScript 的数组类型

JavaScript 数组在 TypeScript 里面分成两种类型，分别是数组（array）和元组（tuple）。

本章介绍数组，下一章介绍元组。

简介

TypeScript 数组有一个根本特征：所有成员的类型必须相同，但是成员数量是不确定的，可以是无限数量的成员，也可以是零成员。

数组的类型有两种写法。第一种写法是在数组成员的类型后面，加上一对方括号。

typescript

```
let arr: number[] = [1, 2, 3];
```

上面示例中，数组 `arr` 的类型是 `number[]` ，其中 `number` 表示数组成员类型是 `number` 。

如果数组成员的类型比较复杂，可以写在圆括号里面。

typescript

```
let arr: (number | string)[];
```

上面示例中，数组 `arr` 的成员类型是 `number|string` 。

这个例子里面的圆括号是必须的，否则因为竖杠 `|` 的优先级低于 `[]` ，TypeScript 会把 `number|string[]` 理解成 `number` 和 `string[]` 的联合类型。

如果数组成员可以是任意类型，写成 `any[]` 。当然，这种写法是应该避免的。

typescript

```
let arr: any[];
```

数组类型的第二种写法是使用 TypeScript 内置的 Array 接口。

typescript

```typescript
let arr: Array<number> = [1, 2, 3];
```

上面示例中，数组 `arr` 的类型是 `Array<number>` ，其中 `number` 表示成员类型是 `number` 。

这种写法对于成员类型比较复杂的数组，代码可读性会稍微好一些。

typescript

```typescript
let arr: Array<number | string>;
```

这种写法本质上属于泛型，这里只要知道怎么写就可以了，详细解释参见《泛型》一章。另外，数组类型还有第三种写法，因为很少用到，本章就省略了，详见《interface 接口》一章。

数组类型声明了以后，成员数量是不限制的，任意数量的成员都可以，也可以是空数组。

typescript

```typescript
let arr: number[];
arr = [];
arr = [1];
arr = [1, 2];
arr = [1, 2, 3];
```

上面示例中，数组 `arr` 无论有多少个成员，都是正确的。

这种规定的隐藏含义就是，数组的成员是可以动态变化的。

typescript

```typescript
let arr: number[] = [1, 2, 3];

arr[3] = 4;
arr.length = 2;

arr; // [1, 2]
```

上面示例中，数组增加成员或减少成员，都是可以的。

正是由于成员数量可以动态变化，所以 TypeScript 不会对数组边界进行检查，越界访问数组并不会报错。

typescript

```typescript
let arr: number[] = [1, 2, 3];
let foo = arr[3]; // 正确
```

上面示例中，变量 `foo` 的值是一个不存在的数组成员，TypeScript 并不会报错。

TypeScript 允许使用方括号读取数组成员的类型。

typescript

```typescript
type Names = string[];
type Name = Names[0]; // string
```

上面示例中，类型 `Names` 是字符串数组，那么 `Names[0]` 返回的类型就是 `string` 。

由于数组成员的索引类型都是 `number` ，所以读取成员类型也可以写成下面这样。

typescript

```typescript
type Names = string[];
type Name = Names[number]; // string
```

上面示例中， `Names[number]` 表示数组 `Names` 所有数值索引的成员类型，所以返回 `string` 。

数组的类型推断

如果数组变量没有声明类型，TypeScript 就会推断数组成员的类型。这时，推断行为会因为值的不同，而有所不同。

如果变量的初始值是空数组，那么 TypeScript 会推断数组类型是 `any[]` 。

typescript

```typescript
// 推断为 any[]
const arr = [];
```

后面，为这个数组赋值时，TypeScript 会自动更新类型推断。

typescript

```typescript
const arr = [];
arr; // 推断为 any[]

arr.push(123);
arr; // 推断类型为 number[]

arr.push("abc");
arr; // 推断类型为 (string|number)[]
```

上面示例中，数组变量 `arr` 的初始值是空数组，然后随着新成员的加入，TypeScript 会自动修改推断的数组类型。

但是，类型推断的自动更新只发生初始值为空数组的情况。如果初始值不是空数组，类型推断就不会更新。

typescript

```
// 推断类型为 number[]
const arr = [123];

arr.push("abc"); // 报错
```

上面示例中，数组变量 `arr` 的初始值是 `[123]` ，TypeScript 就推断成员类型为 `number` 。新成员如果不是这个类型，TypeScript 就会报错，而不会更新类型推断。

只读数组，const 断言

JavaScript 规定， `const` 命令声明的数组变量是可以改变成员的。

typescript

```
const arr = [0, 1];
arr[0] = 2;
```

上面示例中，修改 `const` 命令声明的数组的成员是允许的。

但是，很多时候确实有声明为只读数组的需求，即不允许变动数组成员。

TypeScript 允许声明只读数组，方法是在数组类型前面加上 `readonly` 关键字。

typescript

```
const arr: readonly number[] = [0, 1];

arr[1] = 2; // 报错
arr.push(3); // 报错
delete arr[0]; // 报错
```

上面示例中， `arr` 是一个只读数组，删除、修改、新增数组成员都会报错。

TypeScript 将 `readonly number[]` 与 `number[]` 视为两种不一样的类型，后者是前者的子类型。

这是因为只读数组没有 `pop()` 、 `push()` 之类会改变原数组的方法，所以 `number[]` 的方法数量要多于 `readonly number[]` ，这意味着 `number[]` 其实是 `readonly number[]` 的子类型。

我们知道，子类型继承了父类型的所有特征，并加上了自己的特征，所以子类型 `number[]` 可以用于所有使用父类型的场合，反过来就不行。

typescript

```
let a1: number[] = [0, 1];
let a2: readonly number[] = a1; // 正确

a1 = a2; // 报错
```

上面示例中，子类型 `number[]` 可以赋值给父类型 `readonly number[]` ，但是反过来就会报错。

由于只读数组是数组的父类型，所以它不能代替数组。这一点很容易产生令人困惑的报错。

typescript

```
function getSum(s: number[]) {
  // ...
}

const arr: readonly number[] = [1, 2, 3];

getSum(arr); // 报错
```

上面示例中，函数 `getSum()` 的参数 `s` 是一个数组，传入只读数组就会报错。原因就是只读数组是数组的父类型，父类型不能替代子类型。这个问题的解决方法是使用类型断言 `getSum(arr as number[])` ，详见《类型断言》一章。

注意， `readonly` 关键字不能与数组的泛型写法一起使用。

typescript

```
// 报错
const arr: readonly Array<number> = [0, 1];
```

上面示例中， `readonly` 与数组的泛型写法一起使用，就会报错。

实际上，TypeScript 提供了两个专门的泛型，用来生成只读数组的类型。

typescript

```
const a1: ReadonlyArray<number> = [0, 1];
```

```typescript
const a2: Readonly<number[]> = [0, 1];
```

上面示例中，泛型 `ReadonlyArray<T>` 和 `Readonly<T[]>` 都可以用来生成只读数组类型。两者尖括号里面的写法不一样， `Readonly<T[]>` 的尖括号里面是整个数组（ `number[]` ），而 `ReadonlyArray<T>` 的尖括号里面是数组成员（ `number` ）。

只读数组还有一种声明方法，就是使用“const 断言”。

typescript

```typescript
const arr = [0, 1] as const;

arr[0] = [2]; // 报错
```

上面示例中， `as const` 告诉 TypeScript，推断类型时要把变量 `arr` 推断为只读数组，从而使得数组成员无法改变。

多维数组

TypeScript 使用 `T[][]` 的形式，表示二维数组， `T` 是最底层数组成员的类型。

typescript

```typescript
var multi: number[][] = [
  [1, 2, 3],
  [23, 24, 25],
];
```

上面示例中，变量 `multi` 的类型是 `number[][]` ，表示它是一个二维数组，最底层的数组成员类型是 `number` 。

最后更新: 2023/8/13 15:25

类型系统

元组