数组类型

1. 介绍

在ts中数组有一个根本特征就是各个成员的类型必须相同，成员数量是动态的。

因为数量是动态的，越界访问一个数组成员不会报错,比如xx[4]

2. 写法

- 在数组成员后加方括号[]；对于复杂的成员，则将成员类型加上括号；如果成员是任意的，可以直接写成any[]

```
let arr:number[] = [1, 2, 3];
let arr:(number|string)[];
```

- 使用ts内置的Array接口，对于相对于复杂的类型来说，这样写阅读比较好

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

3. 读取成员类型 不太清楚怎么使用在代码中！！！

ts 允许使用方括号读取数组成员的类型。

```
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。

```
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`。

4. 类型推断

如果数组变量没有声明类型，ts会自动为其推断类型，会根据值的不同推断的类型也不同。

1. 初始化数组为空

如果初始化的数组为空，会推断类型为any[]，后面给数组赋值时，会自动更新数据类型，最终变成联合类型。

2. 初始化数组不为空

初始化时根据成员的值，推断类型，推断后就不能再修改。

5. 只读数组，const断言

在js中，const声明的数组是可以改变的，但是有很多时候需要只读数组的需求，所以ts允许声明只读数组，在数据类型前面加上readonly关键字即可。

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

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

因为arr是一个只读数组，所以不能进行增删改，都会报错。

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

1. 为什么`readonly number[]`是`number[]`的父类型?

这是因为只读数组没有`pop()`、`push()`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。

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。

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

a1 = a2; // 报错
```

2. 可能会出现的问题

当函数的形参为只读数组类型，实参为数组类型，就会报错，因为只读是父类，父类不能赋值给子类。

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

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

getSum(arr) // 报错
```

这个问题的解决方法是使用类型断言`getSum(arr as number[])`，见**类型断言**

3. 注意点

readonly关键字不能和数组的泛型写法一起使用。

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

4. 使用泛型声明只读数组

- 使用ReadonlyArray<T>或者Readonly<T[]>声明。

`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`）。

`Readonly<T[]>`代表整个数组里**都是**T类型的成员。

`ReadonlyArray<T>`代表数组里**存在**T类型的成员。

```

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

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

```

- 使用const断言

as const会告诉ts，推断类型时会把变量arr推断为只读数组，从而使得数组成员无法改变。

```
const arr = [0, 1] as const;
arr[0] = [2]; // 报错
```

6. 多维数组

ts使用`T[][]`的形式，表示二维数组，`T`是最底层数组成员的类型。

```
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`。