TypeScript 类型体操

(TypeScript 类型编程)

Toc

什么是类型体操

TypeScript 是一种 **支持类型编程** 的类型系统。

这不仅意味着我们可以使用基本的类型，还可以对传入的类型参数（泛型）进行各种逻辑运算，从而生成新的类型，这就是类型编程。***由于其相对较高的难度，大家戏称其为 TypeScript 类型体操***。通过类型编程，我们可以在复杂场景下获取精确的类型，从而避免在遇到困难时使用 `any` 类型。

Variables / Constants

- JavaScript

```
let a = 1;
const b = '2';
var c = 'c'
```

- TypeScript Type

```
type A = number
type B = '2'
```

- Variables

```
type A = number | string // 一种或多种类型
let a1: A = 1
let a2: A = 2
let a3: A = '3'
```

- Constants

```
type B = '2' // 具体的值
let b1: B = '2'
let b2: B = '3'
```

Reference

- https://www.TypeScriptlang.org/docs/handbook/2/everyday-types.html

If Else

- JavaScript

```
if(true) {
  // ...
} else {
  // ...
}

const truth = true ? 1 : '2';
```

- TypeScript Type

```
type A = number
type B = string
type C = A extends B ? true : false // 仅支持三元表达式
```

Reference

- https://www.TypeScriptlang.org/docs/handbook/2/conditional-types.html

Loop

- JavaScript

```
for(let i = 0; i < 10; i++) {} // 普通的for循环
for(const item of [1, 2, 3]) {} // for of 循环
for(const key in {a: 1, b: 2}) {} // for in 循环
```

Click the play button to run the code

- TypeScript Type ~~(Object)~~

```
type A = {
  a: number
  b: string
  c: boolean
}
type ALoop = A[keyof A] // A['a'] | A['b'] | A['c']
type AMap = {
  [Key in keyof A]: A[Key]
} // for(const key in {a: 1, b: 2}) {}
```

- TypeScript Type (Array)

```
type B = [number, string, boolean]
type BLoop1 = B extends Array<infer T> ? T : never
type BLoop2 = unknown extends Array<infer T> ? T : never
type Bloop3 = B[keyof B] // B[0] | B[1] | B[2] | property
type Bloop4 = B[number] // B[0] | B[1] | B[2]
```

extends / infer

- `extends` 相当于 js 中的 `===` 。

```
// type
type A = true extends false ? true : false
// js
const a = 1
const b = 1
console.log(a === b)
```

```
true
```

- `infer` 关键词作用是可以完成类型的推导，它只能用于 extends 右侧。

```
// type
type A = [number, string] extends [infer a, string] ? a : n
// js
const [a] = ['abc', '123']
console.log(`a: ${a}`)
```

```
a: abc
```

Challenge 1 - If

```ts
import type { Equal, Expect } from './utils'

type cases = [
  Expect<Equal<If<true, 'a', 'b'>, 'a'>>,
  Expect<Equal<If<false, 'a', 2>, 2>>,
  Expect<Equal<If<boolean, 'a', 2>, 'a' | 2>>,
]

type error = If<null, 'a', 'b'>

type If<C, T, F> = C extends true ? T : F

// function if(c, t, f) {
//   return c ? t : f
// }
```

Challenge 2 - First of Array

```ts
import type { Equal, Expect } from './utils'

type cases = [
  Expect<Equal<First<[3, 2, 1]>, 3>>,
  Expect<Equal<First<[() => 123, { a: string }]>, () => 123>>,
  Expect<Equal<First<[]>, never>>,
  Expect<Equal<First<[undefined]>, undefined>>,
]

type errors = [
  // @ts-expect-error
  First<'notArray'>,
  // @ts-expect-error
  First<{ 0: 'arrayLike' }>,
]


// type First<T extends any[]> = T['length'] extends 0 ? never : T[0]
type First<T extends any[]> = T extends [infer First, ...infer rest] ? First : never

const [a,b,c] = [1,2,3]
```

Challenge 3 - Push

```ts
import type { Equal, Expect } from './utils'

type cases = [
  Expect<Equal<Push<[], 1>, [1]>>,
  Expect<Equal<Push<[1, 2], '3'>, [1, 2, '3']>>,
  Expect<Equal<Push<['1', 2, '3'], boolean>, ['1', 2, '3', boolean]>>,
]

type Push<T extends any[], U> = [...T, U]
```

Challenge 4 - Readonly

```ts
import type { Equal, Expect } from './utils'

type cases = [
  Expect<Equal<MyReadonly<Todo1>, Readonly<Todo1>>>,
]

interface Todo1 {
  title: string
  description: string
  completed: boolean
  meta: {
    author: string
  }
}

type MyReadonly<T> = {
  readonly [P in keyof T]: T[P]
}
```

Challenge 5 - Readonly2

```ts
import type { Alike, Expect } from './utils'

type cases = [
  Expect<Alike<MyReadonly2<Todo1>, Readonly<Todo1>>>,
  Expect<Alike<MyReadonly2<Todo1, 'title' | 'description'>, Expected>>,
  Expect<Alike<MyReadonly2<Todo2, 'title' | 'description'>, Expected>>,
  Expect<Alike<MyReadonly2<Todo2, 'description'>, Expected>>,
]

// @ts-expect-error
type error = MyReadonly2<Todo1, 'title' | 'invalid'>

interface Todo1 {
  title: string
  description?: string
  completed: boolean
}

interface Todo2 {
  readonly title: string
  description?: string
  completed: boolean
}
```

Challenge 6 - Includes

```ts
import type { Equal, Expect } from './utils'

type cases = [
  Expect<Equal<Includes<['Kars', 'Esidisi', 'Wamuu', 'Santana'], 'Kars'>, true>>,
  Expect<Equal<Includes<['Kars', 'Esidisi', 'Wamuu', 'Santana'], 'Dio'>, false>>,
  Expect<Equal<Includes<[1, 2, 3, 5, 6, 7], 7>, true>>,
  Expect<Equal<Includes<[1, 2, 3, 5, 6, 7], 4>, false>>,
  Expect<Equal<Includes<[1, 2, 3], 2>, true>>,
  Expect<Equal<Includes<[1, 2, 3], 1>, true>>,
  Expect<Equal<Includes<[{}], { a: 'A' }>, false>>,
  Expect<Equal<Includes<[boolean, 2, 3, 5, 6, 7], false>, false>>,
  Expect<Equal<Includes<[true, 2, 3, 5, 6, 7], boolean>, false>>,
  Expect<Equal<Includes<[false, 2, 3, 5, 6, 7], false>, true>>,
  Expect<Equal<Includes<[{ a: 'A' }], { readonly a: 'A' }>, false>>,
  Expect<Equal<Includes<[{ readonly a: 'A' }], { a: 'A' }>, false>>,
  Expect<Equal<Includes<[1], 1 | 2>, false>>,
  Expect<Equal<Includes<[1 | 2], 1>, false>>,
  Expect<Equal<Includes<[null], undefined>, false>>,
  Expect<Equal<Includes<[undefined], null>, false>>,
]

type Includes<T extends readonly any[], U> = T extends [infer F, ...infer R] ?
  Equal<F, U> extends true ? true : Includes<R, U> : false
```