

高级类型

2021年10月8日 18:45

1. 索引类型

```
``js
let obj_18 = {
  a: 1,
  b: 2,
  c: 3
}
// 获取某个对象的若干个属性值组成数组
function getValues(obj: any, keys: string[]) {
  return keys.map(key => obj[key]);
}
console.log(getValues(obj_18, ['a', 'b'])); // [1, 2]
console.log(getValues(obj_18, ['e', 'f'])); // [undefined, undefined]
``
```

当获取对象不存在的键值时，ts编译器并不会报错。

我们希望 keys 里元素是对象的属性，
否则报错。

通过 keyof 泛型约束 T[k] 的搭配使用实现索引类型。

1. 三个基本概念

```
``js
// keyof T —— T的公共属性的字面量属性的联合类型
interface Obj_18 {
  a: number,
  b: string
}
let key_18: keyof Obj_18;
// T[k] 对象T的属性k所代表的类型。
let value_18: Obj_18['a']; // value_18 是 number类型

// 泛型约束: T extends U 通过继承某些变量，获得属性
``
```

2. 改造

```
``js
// 获取某个对象的若干个属性值组成数组
function getValues18<T, K extends keyof T>(obj: T, keys: K[]): T[K][] {
  return keys.map(key => obj[key]);
}
getValues18(obj_18, ['c', 'e']); // 不能将类型 "e" 分配给类型 "a" | "b" | "c"
``
```

2. 映射类型

从一个旧的类型生成一个新的类型。本质上是一个泛型接口，内部通常用到了索引类型。

// Readonly 是一个 TS 内置的泛型接口

2.1 type ReadonlyObj_19 = Readonly<Obj_19>

```
1 import { type } from "jquery";
2 "ReadonlyObj_19"已声明，但从未使用过。 ts(6196)
3 interface ReadonlyObj_19 = {
4   a: readonly number;
5   b: readonly string;
6   c: readonly boolean;
7 }
8
9 // Re 快速修复 (Ctrl+)
10 type ReadonlyObj_19 = Readonly<Obj_19>
```

2.2 type PickObj_19 = Pick<Obj_19, 'a' | 'b'>

```
3 interface PickObj_19 = Pick<Obj_19, 'a' | 'b'>
4 type PickObj_19 = {
5   a: number;
6   b: string;
7 }
8
9 // Re
10 "PickObj_19"已声明，但从未使用过。 ts(6196)
11 快速修复 (Ctrl+)
12 type PickObj_19 = Pick<Obj_19, 'a' | 'b'>
```

2.3 type RecodeObj_19 = Record<'y' | 'x', Obj_19>;

```
1 C. boolean
2 }
3 "RecodeObj_19"已声明，但从未使用过。 ts(6196)
4 // Re
5 type RecodeObj_19 = {
6   y: Obj_19;
7   x: Obj_19;
8 }
9
10 快速修复 (Ctrl+)
11 type RecodeObj_19 = Record<'y' | 'x', Obj_19>;
```

```
/**
 * Make all properties in T readonly
 */
type Readonly<T> = {
  readonly [P in keyof T]: T[P];
};

/**
 * From T, pick a set of properties whose keys are in the union K
 */
type Pick<T, K extends keyof T> = {
  [P in K]: T[P];
};

/**
 * Construct a type with a set of properties K of type T
 */
type Record<K extends any, T> = {
  [P in K]: T;
};
```

3. 条件类型

```
// 条件类型 T extends U ? X : Y
import { type } from "jquery";
type TypeName<T> = T extends string ? string :
```

```
T extends number ? number :  
T extends boolean ? boolean :  
T extends undefined ? undefined :  
T extends Function ? Function :  
object;  
  
type T1 = TypeName<'1'>;  
let s_21: T1 = '1';  
type T2 = TypeName<{}>;  
let o_21: T2 = {};  
type T3 = TypeName<() => ''>;  
let f_21: T3 = function() {};
```