



下载APP

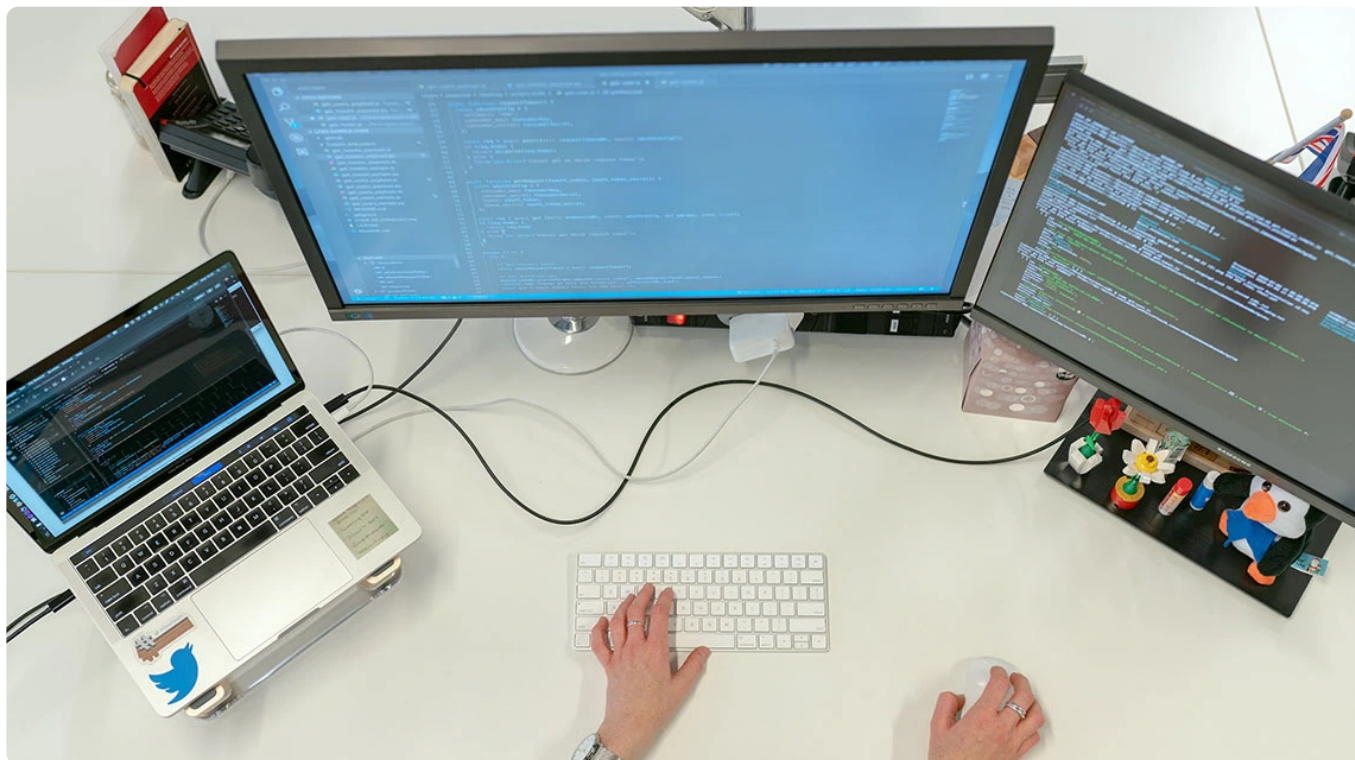


## 加餐02 | 深入TypeScript

2021-12-03 大圣

《玩转Vue 3全家桶》

课程介绍 &gt;

**讲述：大圣**

时长 10:06 大小 9.25M



你好，我是大圣。

在讲组件化的进阶开发篇之前，我想在全家桶实战篇的最后，用一讲的篇幅，来专门聊一下 TypeScript。希望你在学完这一讲之后，能对 TypeScript 有一个全面的认识。


另外，今天我会设置很多实战练习，一边阅读一边敲代码的话，学习效果更好。而且，这次加餐中的全部代码都是可以在线完成的，建议你打开 [这个链接](#)，把下面的每行代码都跟着敲一遍。



## TypeScript 入门


对于 TypeScript，你首先要了解的是，TypeScript 可以在 JavaScript 的基础上，对变量的数据类型加以限制。TypeScript 中最基本的数据类型包括布尔、数字、字符串、null、undefined，这些都很好理解。

在下面的代码中，我们分别定义了这几个数据类型的变量，你能看到，当我们把 number 类型的变量 price 赋值字符串时，就会报错，当我们把数组 me 的第一个元素 me[0] 的值修改为数字时，也会报错。

 复制代码


```
1 let courseName:string = '玩转Vue 3全家桶'
2 let price:number = 129
3 price = '89' //类型报错
4 let isOnline:boolean = true
5 let courseSales:undefined
6 let timer:null = null
7 let me:[string,number] = ["大圣",18]
8 me[0] = 1 //类型报错
```

当你不确定某个变量是什么类型时，你可以使用 any 作为这个变量的类型。你可以用 any 标记任何属性，可以修改任何数据，访问任何方法也不会报错。也就是说，在 TypeScript 中，当你把变量的类型标记为 any 后，这个变量的使用就和 JavaScript 没啥区别了，错误只会在浏览器里运行的时候才会提示。

 复制代码

```
1 let anything
2 let anyCourse :any = 1
3 anyCourse = 'xx'
4 console.log(anyCourse.a.b.c)
```

然后我们可以使用 enum 去定义枚举类型，这样可以把类型限制在指定的场景之内。下面的代码中我们可以把课程评分限制在好、非常好和嘎嘎好三个值之内。

 复制代码

```
1 enum 课程评分 {好,非常好,嘎嘎好}
2 console.log(课程评分['好']===0)
3 console.log(课程评分[0]=== '好')
4 let scores = [课程评分['好'], 课程评分['嘎嘎好'], 课程评分['非常好']]
5
```

然后我们可以通过学到的这些基础类型，通过组合的方式组合出新的类型，最常见的组合方式就是使用 `|` 实现类型联合。下面的代码中我们定义 `course1` 变量的类型为字符串或者数字，赋值为这两个类型都不会报错，还可以用来限制变量只能赋值为几个字符串的一个，`score` 的取值只能是代码中三个值之一。

[复制代码](#)


```
1 let course1 : string|number = '玩转vue 3'
2 course1 = 1
3 course1 = true // 报错
4
5 type courseScore = '好' | '非常好' | '嘎嘎好'
6 let score1 :courseScore = '好'
7 let score2 :courseScore = '一般好' // 报错
```

通过 `interface` 接口可以定义对象的类型限制。下面代码中我们定义了极客时间课程的类型，课程名是字符串，价格使用 `number[]` 语法定义类型为数字组成的数组，讲师头像是 `string` 或者 `boolean`，并且通过 `?` 设置为可选属性，课程地址使用 `readonly` 设置为只读属性，如果对课程地址进行修改就会报错。

[复制代码](#)


```
1 interface 极客时间课程 {
2     课程名字:string,
3     价格:number[],
4     受众:string,
5     讲师头像?:string|boolean,
6     readonly 课程地址:string
7 }
8 let vueCourse: 极客时间课程 = {
9     课程名字:'玩转Vue 3全家桶',
10    价格:[59,'139'],
11    讲师头像:false,
12    课程地址:"time.geekbang.org"
13 }
14 vueCourse.课程地址 = 'e3.shengxinjing.cn' // 报错
```

然后我们学一下函数的类型限制。其实函数的定义，参数和返回值本质上也是变量的概念，都可以进行类型的定义。下面的代码中我们定义了参数 `x` 和 `y` 是数字，返回值也是数字的 `add` 函数，定义好参数和返回值类型，函数的类型自然也就确定了。

 复制代码


```
1 function 函数名(参数:参数类型):返回值类型{} //大致语法
2 function add(x: number, y: number): number {
3     return x + y;
4 }
5 add(1, 2);
6
```

我们也可以使用变量的方式去定义函数，直接使用 (参数类型) => 返回值类型的语法去定义 add1 的变量类型，但是这样写出来的代码可读性稍差一些，我更建议你使用 type 或者 interface 关键字去定义函数的类型。下面代码中的 addType 和 addType1 都是很好的定义函数类型的方式：

 复制代码

```
1 let add1:(a:number,b:number)=>number = function(x: number, y: number): number
2     return x + y;
3 }
4 type addType = (a:number,b:number)=>number
5 let add2:addType = function(x: number, y: number): number {
6     return x + y;
7 }
8
9 interface addType1{
10     (a:number,b:number):number
11 }
12 let add3:addType1 = function(x: number, y: number): number {
13     return x + y;
14 }
```

如果你的函数本来就支持多个类型的参数，下面的代码中 reverse 函数既支持数字也支持字符串。**我们的要求是如果参数是数字，返回值也要是数字，参数是字符串返回值也只能是字符串**，所以参数和返回值都用 number|string 就没法精确地限制这个需求。我们需要使用函数重载的方式，定义多个函数的输入值和返回值类型，更精确地限制函数的类型。我们可以在 [Vue 3 的源码](#) 看到 Vue 3 中 ref 函数的重载写法：

 复制代码

```
1 function reverse(x: number): number
2 function reverse(x: string): string
3 function reverse(x: number | string): number | string | void {
4     if (typeof x === 'number') {
5         return Number(x.toString().split('').reverse().join(''));
6     }
7 }
```

```
6     } else if (typeof x === 'string') {
7         return x.split('').reverse().join('');
8     }
9 }
```

这样 TypeScript 里如何限制一个变量和函数类型，我们就大致入门了。这时候你肯定还有个疑问，**日常开发中有很多浏览器上的变量和属性，这些怎么限制类型呢？**

关于宿主环境里的类型，TypeScript 全部都给我们提供了，我们可以直接在代码中书写：Window 是 window 的类型，HTMLElement 是 dom 元素类型，NodeList 是节点列表类型，MouseEvent 是鼠标点击事件的类型.....关于更多 TypeScript 的内置类型，你可以在 [🔗TypeScript 的源码](#)中看到：

[📄 复制代码](#)

```
1 let w:Window = window
2 let ele:HTMLElement = document.createElement('div')
3 let allDiv: NodeList = document.querySelectorAll('div')
4
5 ele.addEventListener('click',function(e:MouseEvent){
6     const args:IArguments = arguments
7     w.alert(1)
8     console.log(args)
9 },false)
10
```

除了浏览器的 API，我们还会用到很多第三方框架，比如 Vue、Element3 等等，这些框架现在都提供了完美的类型可以直接使用。在 [🔗第 18 讲](#)中我们使用下面的代码 Vue 导出的 Ref 来限定数据是 ref 包裹的响应式数据：

[📄 复制代码](#)

```
1 import { ref ,Ref} from 'vue'
2 interface Todo{
3     title:string,
4     done:boolean
5 }
6 let todos:Ref = ref([{title:'学习Vue',done:false}])
7
8
```

## 泛型

那么聊完上面的内容，你就已经能使用 TypeScript 实现很多项目的开发，把所有变量和函数出现的地方都定义好类型，就可以在编译阶段提前规避出很多报错。然而 TypeScript 的能力可不止于此，**TypeScript 可以进行类型编程，这会极大提高 TypeScript 在复杂场景下的应用场景。**

然后我们来看一下 TypeScript 中的泛型，这也是很多同学觉得 TypeScript 很难的最大原因。

首先我们看下面的代码，我们定一个 identity0 函数，这个函数逻辑非常简单，就是直接返回参数，那么**我们怎么确定返回值的类型呢？**

因为输入值可以是任意属性，所以我们只能写出 identity0 这个函数，参数和返回值类型都是 any，但是明显不能满足我们的需求。我们需要返回值的类型和参数一致，所以我们在函数名之后使用 <> 定一个泛型 T，你可以理解这个 T 的意思就是给函数参数定义了一个类型变量，会在后面使用，相当于【**type T = arg 的类型**】，返回值使用 T 这个类型就完成了这个需求。

[复制代码](#)

```
1 function identity0(arg: any): any {  
2     return arg  
3 }  
4 // 相当于type T = arg的类型  
5 function identity<T>(arg: T): T {  
6     return arg  
7 }  
8 identity<string>('玩转vue 3全家桶') // 这个T就是string，所以返回值必须得是string  
9 identity<number>(1)
```

有了泛型之后，我们就有了把函数参数定义成类型的功能，我们就可以实现类似高阶函数的类型函数。下面的代码中我们使用 keyof 语法获得已知类型 VueCourse5 的属性列表，相当于 'name' | 'price'：

[复制代码](#)

```
1 interface VueCourse5 {  
2     name:string,  
3     price:number
```



```

4  }
5  type CourseProps = keyof VueCourse5 // 只能是name和price选一个
6  let k:CourseProps = 'name'
7  let k1:CourseProps = 'p' // 改成price

```

keyof 可以帮助我们拆解已有类型，下一步我们需要使用 extends 来实现类型系统中的条件判断。我们定义类型函数 ExtendsType，接受泛型参数 T 后，通过判断 T 是不是布尔值来返回不同的类型字符串，我们就可以通过 ExtendsType 传入不同的参数去返回不同的类型。

[复制代码](#)

```

1  // T extends U ? X : Y 类型三元表达式
2
3  type ExtendsType<T> = T extends boolean ? "重学前端" : "玩转Vue 3"
4  type ExtendsType1 = ExtendsType<boolean> // type ExtendsType1='重学前端'
5  type ExtendsType2 = ExtendsType<string> // type ExtendsType2='玩转Vue 3'

```

extends 相当于 TypeScript 世界中的条件语句，然后 in 关键字可以理解为 TypeScript 世界中的遍历。下面的代码中我们通过 k in Courses 语法，相当于遍历了 Courses 所有的类型作为 CourseObj 的属性，值的类型是 number。

[复制代码](#)

```

1  type Courses = '玩转Vue 3' | '重学前端'
2  type CourseObj = {
3      [k in Courses]: number // 遍历Courses类型作为key
4  }
5  // 上面的代码等于下面的定义
6  // type CourseObj = {
7  //     玩转Vue 3: number;
8  //     重学前端: number;
9  // }
10

```

学完上面的语法，你就能完全搞懂 [第 18 讲](#) 里的 getProperty 函数。限制函数第二个参数只能是第一个参数的属性，并且返回值的类型，最后我们传递不存在的属性时，TypeScript 就会报错。

[复制代码](#)

```

1  // K extends keyof T 限制K的类型必须是T的属性之一

```

```
2 // T[K]是值得类型
3 function getProperty<T, K extends keyof T>(o: T, name: K): T[K] {
4     return o[name]
5 }
6 const coursePrice: CourseObj = {
7     "玩转Vue 3": 129,
8     "重学前端": 129
9 }
10 getProperty(coursePrice, '玩转Vue 3')
11 getProperty(coursePrice, '不学前端') // 报错
```

然后我再给你讲解最后一个关键字 `infer`。`<T>` 让我们拥有了给函数的参数定义类型变量的能力，`infer` 则是可以在 `extends` 之后的变量设置类型变量，更加细致地控制类型。下面的代码中我们定义了 `ReturnType` 类型函数，目的是返回传入函数的返回值类型。`infer P` 的意思就是泛型 `T` 是函数类型，并且这个函数类型的返回类型是 `P`。

[复制代码](#)

```
1 type Foo = () => CourseObj
2
3 // 如果T是一个函数，并且函数返回类型是P就返回P
4 type ReturnType<T> = T extends ()=>infer P ?P:never
5 type Foo1 = ReturnType<Foo>
```

## 实战练习

有了上面的基础后，我们来几个实战的练习。以下所有的练习都可以在代码最后找到答案，我建议你一定要自己实现一遍才能有最多的收获。

代码地址：<https://www.typescriptlang.org/docs/handbook/utility-types.html>

下面的代码中，我们首先定义类型 `Todo`，有 `title`、`desc` 和 `done` 三个属性：

[复制代码](#)

```
1 interface Todo {
2     title: string
3     desc: string
4     done: boolean
5 }
6
```



首先第一题是，我们需要实现类型函数 `Partial1`，返回的类型是 `Todo` 所有的属性都变成可选项。

[复制代码](#)

```
1 type partTodo = Partial1<Todo>
2 // 和下面类型一直，鼠标移动到partTodo变量上也能看到
3 // type partTodo = {
4 //     title?: string | undefined;
5 //     desc?: string | undefined;
6 //     done?: boolean | undefined;
7 // }
```

这一题的答案见下面的代码，使用 `K in keyof T` 遍历所有 `T` 的属性后，使用 `?` 标记为可选属性。

[复制代码](#)


```
1 type Partial1<T> = {
2     [K in keyof T]?: T[K]
3 }
```

TypeScript 中还有很多类似的函数，包括 `Pick`、`Omit`、`Diff` 等函数，你都可以自行实现一遍，更多工具类型函数你可以移步 [TypeScript 官方文档](#)。你也可以结合下面的代码工具函数的实现，到留言区中讨论一下分别实现了什么功能。

[复制代码](#)


```
1 type Exclude1<T, K> = T extends K ? never : K
2 type Pick1<T, K extends keyof T> = {
3     [P in K]: T[P]
4 }
5 type Concat1<T extends any[], U extends any[]> = [...T, ...U]
```

最后我们再来一个实战的练习，在实际项目开发中除了 JavaScript、浏览器和第三方框架的类型，还有一个很重要的场景就是后端发挥的数据类型。我们需要根据开发文档去定义好每个请求的类型，在下面的代码中，`request` 作为发送请求的函数，可以传递 `url` 是字符串。那我们该如何定义 `Interface API`，使其能够限制 `request` 只能有 `bug` 和 `comment` 两个请求地址，并且 `comment` 请求的参数中 `message` 是必传项呢？

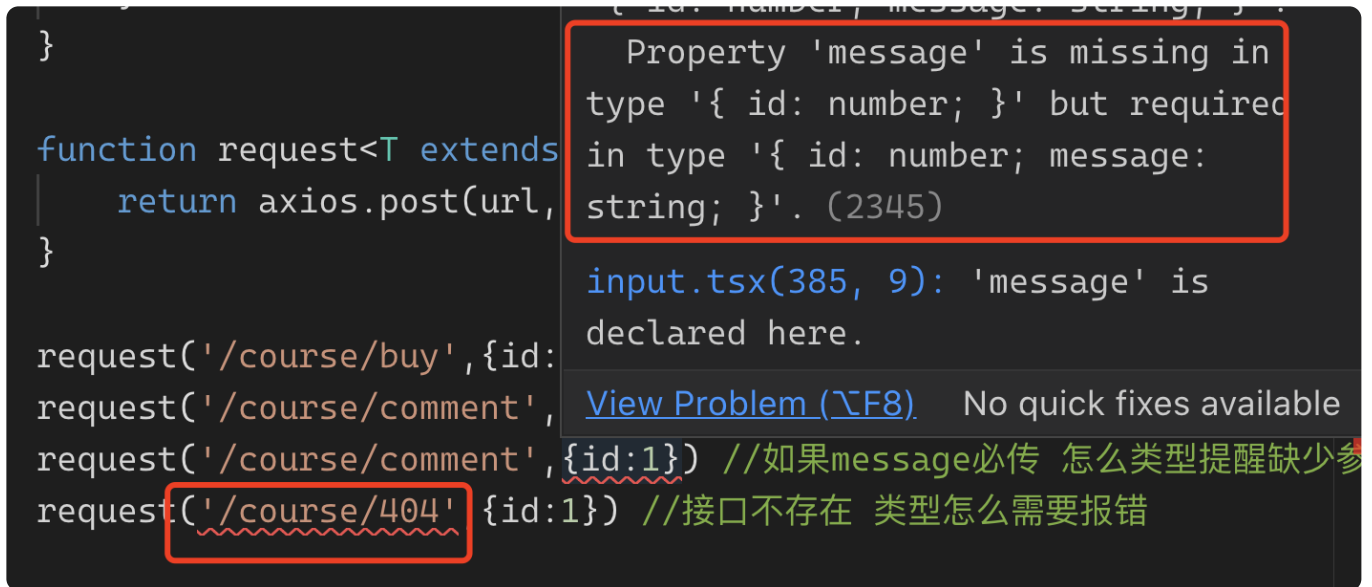
 复制代码

```
1 import axios from 'axios'
2
3 function request(url:string,obj:any){
4     return axios.post(url,obj)
5 }
6 interface Api{
7
8 }
9 request('/course/buy',{id:1})
10 request('/course/comment',{id:1,message:'嘎嘎好看'})
11 request('/course/comment',{id:1}) //如果message必传 怎么类型提醒缺少参数
12 request('/course/404',{id:1}) //接口不存在 类型怎么需要报错
13
```

记得要先尝试自己实现一下，答案就在下面的代码中。在 API 类型中，我们定义了 `buy` 和 `comment` 两个属性，分别设置了当前请求所需要的参数都是必选项。然后我们通过 `request` 中使用泛型 `T` 限制 `url`，通过 `Api[T]` 限制传递的参数，这样我们就得到了下面的报错示意图，在编译阶段就能通知你缺少 `message` 属性，并且 `404` 请求不存在，这可以极大提高我们开发的体验和效率。

 复制代码

```
1 import axios from 'axios'
2 interface Api{
3     '/course/buy':{
4         id:number
5     },
6     '/course/comment':{
7         id:number,
8         message:string
9     }
10 }
11
12 function request<T extends keyof Api>(url:T,obj:Api[T]){
13     return axios.post(url,obj)
14 }
15
16 request('/course/buy',{id:1})
17 request('/course/comment',{id:1,message:'嘎嘎好看'})
18 request('/course/comment',{id:1}) //如果message必传 怎么类型提醒缺少参数
19 request('/course/404',{id:1}) //接口不存在 类型怎么需要报错
20
```



更多类型的练习，你可以访问 [type-challenges](#) 这个项目自行尝试。现在你再去看项目或者框架源码中的 TypeScript，是不是就没有那么晦涩了呢。

## 总结

今天想聊的 TypeScript 就结束啦，总结一下我们今天学到的内容吧。

首先我们学习了 TypeScript 的基本类型，包括数字字符串等等，然后我们可以通过这些基础类型组合出复杂的类型组合，并且可以通过 type 和 interface 关键字定义复杂对象的类型和函数的类型。

然后浏览器的相关变量和 API 类型 TypeScript 都已经内置了，包括 HTMLElement、MouseEvent 等等，第三方框架的类型我们可以直接导入使用，Vue 中的 Ref 类型我们就会经常用来定义 ref 函数包裹的响应式数据。

接着我们学习了 TypeScript 进阶中最重要的概念：泛型。通过泛型我们可以在函数内部把类型变成变量使用，并且通过 keyof、in、extends、infer 等关键字组合出复杂的类型函数，可以更加精确地组合现有类型。

最后我们通过定义前后端的接口类型的案例，演示了在实战中我们如何通过类型系统提高联调和开发的体验。

有了这些 TypeScript 的知识储备，你才能更好地在 Vue 项目中使用 TypeScript。由于 JSX 的本质就是 JavaScript，所以 TypeScript 的诞生也给了 JSX 更好的类型推导，这是

JSX 相比于 Template 的另外一个优势。关于 Vue 和 TypeScript 开发组件的内容，我们下一讲开始全部使用 TypeScript 来实现。


## 思考题

最后，留一个思考题：我们实现的 Partial1 可以把类型的属性变成可选的，如果传递的类型是嵌套很多层的，如何实现 Partial1 的递归版本，才能把所有嵌套的属性都变成可选的呢？

欢迎你在评论区分享你的答案，也欢迎你把这一讲的内容分享给你的同事和朋友们，我们下一讲再见。

分享给需要的人，Ta订阅后你可得 **20 元现金奖励**

 生成海报并分享

 赞 6  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 19 | 实战痛点5：如何打包发布你的Vue 3应用？

更多课程推荐

# 跟月影学可视化

## 系统掌握图形学与可视化核心原理

月影

奇虎 360 奇舞团团长

可视化 UI 框架 SpriteJS 核心开发者



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

### 精选留言 (8)

[写留言](#)

山雨

2021-12-03

大圣老师，实战篇代码啥时候更新

展开 ∨



👍 1



cwang

2021-12-03

有个地方挺神奇的哈，不打印你是不会理解这段代码的。

```
enum 课程评分 { 好, 非常好, 嘎嘎好 }  
console.log(课程评分['好'] === 0) // true  
console.log(课程评分[0] === '好') // true...
```

展开 ∨



👍 1



韩棒

2021-12-03

大圣老师 Ref在使用的時候有一個“Ref”僅表示類型，但在此處卻作為值使用-ts。是需要配置tsconfig.json嗎？

展开 ∨



**cwang**

2021-12-03

不建議用中文變量名，其中一個原因是手敲的時候，需要來回轉換中英文輸入法。😁



**cwang**

2021-12-03

So Cool !

展开 ∨



**南山**

2021-12-03

```
type RecursivePartial<T> = {  
  [P in keyof T]?:  
    T[P] extends (infer U)[] ? RecursivePartial<U>[] :  
    T[P] extends object ? RecursivePartial<T[P]> :  
    T[P];...
```

展开 ∨



**費城的二鵬**

2021-12-03

太實用了，周末要親手試試，避免用any，減少出錯可能。



**海闊天空**

2021-12-03

如果可以的話，在項目中引用ts還是很有必要的。對項目的規範和要求也相對提升。

