为什么你非常不适应 TypeScript

前言

在群里看到一些问题和言论：为什么你们这么喜欢“类型体操”？为什么我根本学不下去 TypeScript？我最讨厌那些做类型体操的了；为什么我学了没过多久马上又忘了？

有感于这些问题，我想从最简单的一个角度来切入介绍一下 TypeScript，并向大家介绍并不是只要是个类型运算就是体操。并在文中介绍一种基本思想作为你使用类型系统的基本指引。

引子

我将从一个相对简单的 API 的设计过程中阐述关于类型的故事。在这里我们可以假设我们现在是一个工具的开发者，然后我们需要设计一个 API 用于从对象中拿取指定的一些 key 作为一个新的对象返回给外面使用。

垃圾 TypeScript

一个人说：我才不用什么破类型，我写代码就是要没有类型，我就是要随心所欲的写。然后写下了这段代码。

```
1 declare function pick(target: any, ...keys: any): any
```

他的用户默默的写下了这段代码：

```
1 pick(undefined, 'a', 1).b
```

写完运行，发现问题大条了，控制台一堆报错，接口数据也提交不上去了，怎么办呢？

刚学 TypeScript

一个人说：稍微检查一下传入类型就好了，别让人给我乱传参数就行。

```
1 declare function pick(target: Record<string, unknown>, ...keys: string[]):
  unknown
```

很好，上面的问题便不复存在了，API 也是基本可用的了。但是！当对象复杂的时候，以及字段并不是短单词长度的时候就会发现了一个没解决的问题。

```
pick({ abcdefghijkl: '123' }, 'abcdefghikjl')
```

从肉眼角度上，我们很难发现这前后的不一致，所以我们为什么要让调用方的用户自己去 check 自己的字段有没有写对呢？

不就 TypeScript

一个人说：这还不简单，用个泛型加 keyof 不就行了。

```
declare function pick<
  T extends Record<string, unknown>
>(target: T, ...keys: (keyof T)[]): unknown
```

我们又进一步解决的上面的问题，但是！还是有着相似的问题，虽然我们不用检查 keys 是不是传入的是一个正确的值了，但是我们实际上对返回的值也存在一个类似的问题。

```
pick({ abcdefghijkl: '123' }, 'abcdefghijkl').abcdefghikjl
```

- 一点小小的拓展
- 在这里我们看起来似乎是一个很简单的功能，但实际上蕴含着一个比较重要的信息。
- 为什么我们之前的方式都拿不到用户传入进来的类型信息呢？是有原因的，当我们设计的 API 的时候，前面的角度是从，如何校验类型方向进行的思考。
- 而这里是尝试去通过约定好的一种规则，通过 TypeScript 的隐式类型推断获得到传入的类型，再通过约定的规则转化出一种新的类型约束来对用户的输入进行限制。

算算 TypeScript

一个人说：好办，算出来一个新的类型就好了。

```
declare function pick<
  T extends Record<string, unknown>,
  Keys extends keyof T
>(target: T, ...keys: Keys[]): {
  [K in Keys]: T[K]
```

```
}
```

到这里已经是对类型的作用有了基础的了解了，能写出来符合开发者所能接受的类型相对友好的代码了。我们可以再来思考一些更特殊的情况：

```
// 输入了重复的 key
pick({ a: '' }, 'a', 'a')
```

完美 TypeScript

到这里，我们便是初步开始了类型“体操”。但是在本篇里，我们不去分析它。

```
export type L2T<L, LAlias = L, LAlias2 = L> = [L] extends [never]
  ? []
  : L extends infer LItem
    ? [LItem?, ...L2T<Exclude<LAlias2, LItem>, LAlias>]
    : never

declare function pick<
  T extends Record<string, unknown>,
  Keys extends L2T<keyof T>
>(target: T, ...keys: Keys): Pick<T, Keys[number] & keyof T>

const x0 = pick({ a: '1', b: '2' }, 'a')
console.log(x0.a)
// @ts-expect-error
console.log(x0.b)

const x1 = pick({ a: '1', b: '2' }, 'a', 'a')
//                                  ^^^^^^^^
// TS2345: Argument of type '["a", "a"]' is not assignable to parameter of type '["a"?, "b"?] | ["b"?, "a"?]'.
//   Type '["a", "a"]' is not assignable to type '["a"?, "b"?]'.
//     Type at position 1 in source is not compatible with type at position 1 in target.
//       Type '"a"' is not assignable to type '"b"'.
```

一个相对来说比较完美的 pick 函数便完成了。

总结

我们再来回到我们的标题吧，从我对大多数人的观察来说，很多的人开始来使用 TypeScript 有几种原因：

- 看到大佬们都在玩，所以自己也想来“玩”，然后为了过类型校验而去写
- 看到一些成熟的项目在使用 TypeScript，想参与贡献，参与过程中为了让类型通过而想办法去解决类型报错
- 公司整体技术栈采用的是 TypeScript，要用 TypeScript 进行业务编写，从而为了过类型检查和 review 而去解决类型问题

诸如此类的问题还有很多，我将这种都划分为「为了解决类型检查的问题」而进行的类型编程，这也是大多数人为什么非常不适应 TypeScript，甚至不喜欢他的一个原因。这其实对学习 TypeScript 并不是一个很好的思路，在这里我觉得我们需要站在设计者的角度去对类型系统进行思考。我觉得有以下几个角度：

- 类型检查到位
- 类型提示友好
- 类型检查严格
- 扩展性十足

我们如果站在这几个角度对我们的 API 进行设计，我们可以发现，开发者能够很轻松的将他们需要的代码编写出来，而尽量不用去翻阅文档，查找 example。

希望通过我的这篇分享，大家能对 TypeScript 多一些理解，并参与到生态中来，守护我们的 JavaScript。

理性探讨，说什么屎不是屎的，嘴巴臭可以不说话的。

没谁逼着你一定要写最后一种层次的代码，能力不足可以学啊，不喜欢可以不学啊，能达到倒数第二个就已经很棒啊。

最后一种只是给大家看看 TypeScript 的一种可能，而不是说你应该这么做的。