TypeScript 练习题(第二弹)

2020-10-13

TypeScript 的学习资料非常多，其中也不乏很多优秀的文章和教程。但是目前为止没有一个我特别满意的。原因有：

* 它们大多数没有一个清晰的主线，而是按照 API 组织章节的，内容在**逻辑上**比较零散。
* 大多是“讲是什么，怎么用“，而不是”讲为什么，讲原理“。
* 大多数内容比较枯燥，趣味性比较低。都是干巴巴的文字，没有图片，缺乏能够引起强烈共鸣的例子。

因此我的想法是做一套不同市面上大多数的 TypeScript 学习教程。以人类认知的角度思考问题，学习 TypeScript，通过通俗易懂的例子和图片来帮助大家建立 TypeScript 世界观。

系列安排：

* [上帝视角看 TypeScript](https://lucifer.ren/blog/2020/08/04/ts-internal/)
* [TypeScript 类型系统](https://lucifer.ren/blog/2020/08/15/ts-type-system/)
* [types 和 @types 是什么？](https://lucifer.ren/blog/2020/08/21/ts-type/)
* [你不知道的 TypeScript 泛型（万字长文，建议收藏）](https://lucifer.ren/blog/2020/06/16/ts-generics/)
* [TypeScript 配置文件该怎么写？](https://lucifer.ren/blog/2020/08/24/ts-config/)
* TypeScript 是如何与 React，Vue，Webpack 集成的？
* [TypeScript 练习题(第一弹)](https://lucifer.ren/blog/2020/09/27/ts-exercises/)

目录将来可能会有所调整。

注意，我的系列文章基本不会讲 API，因此需要你有一定的 TypeScript 使用基础，推荐两个学习资料。

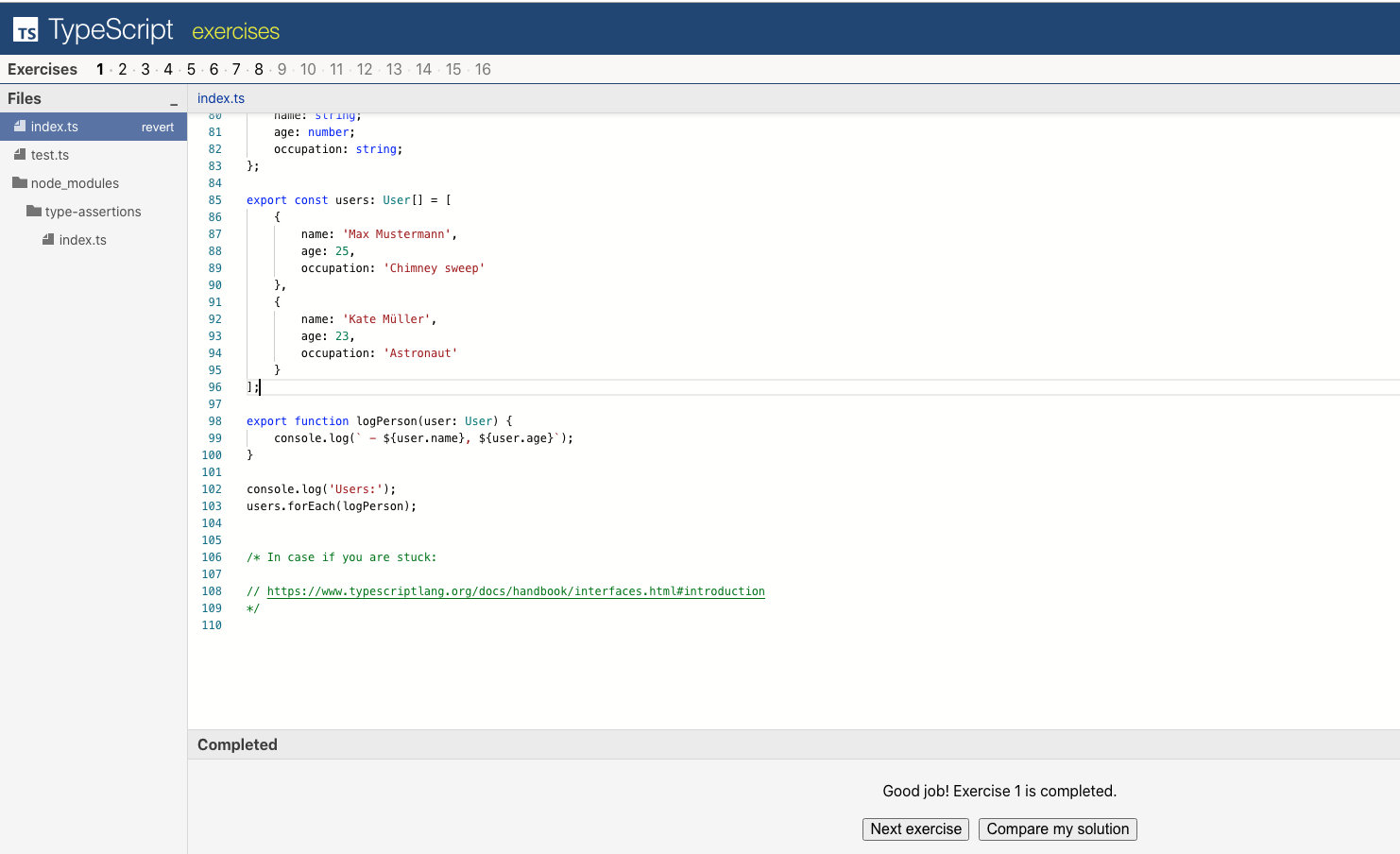
* [深入理解 TypeScript](https://jkchao.github.io/typescript-book-chinese/)
* [TypeScript 官方文档](https://www.typescriptlang.org/docs/home)

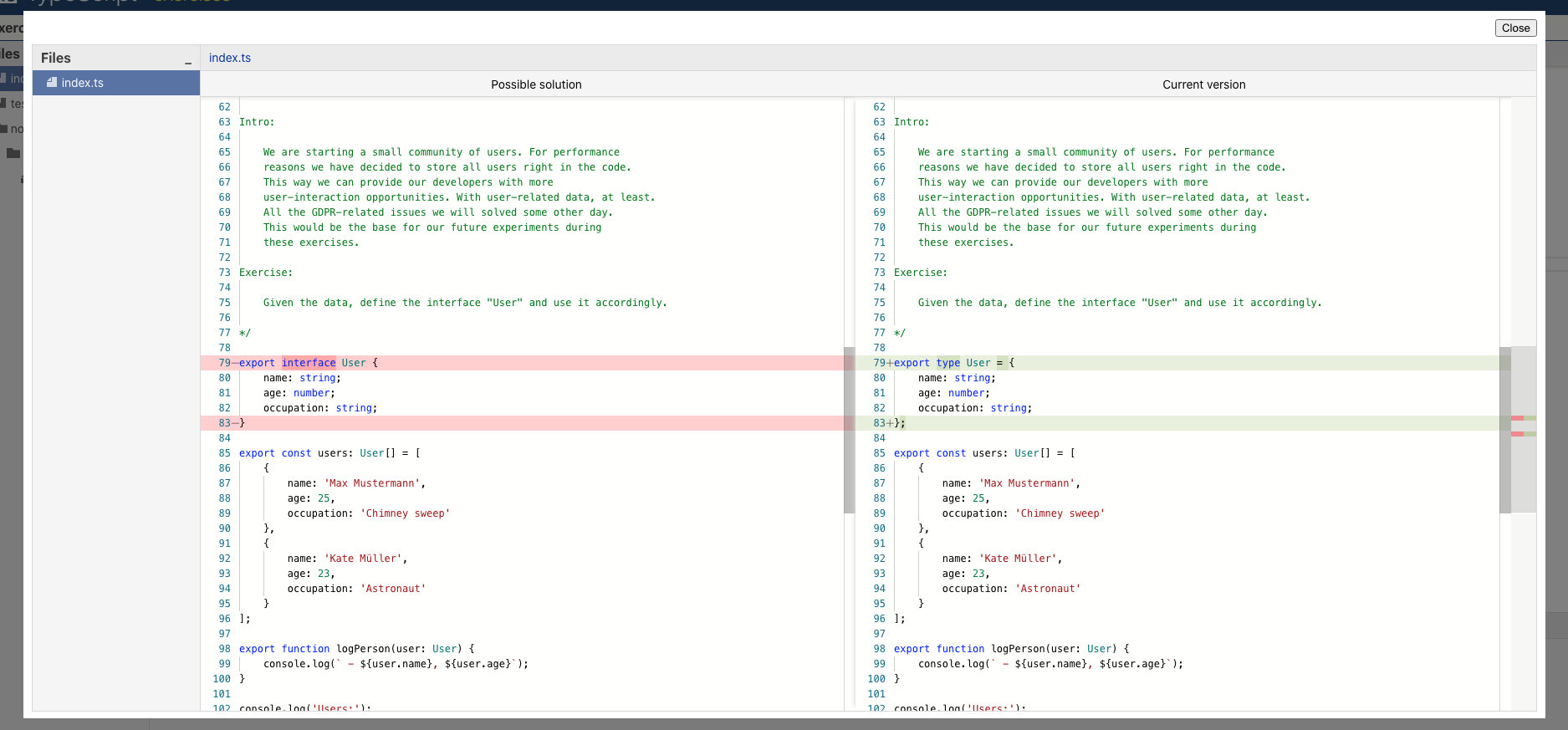
结合这两个资料和我的系列教程，掌握 TypeScript 指日可待。

接下来，我们通过几个方面来从宏观的角度来看一下 TypeScript。

## 前言

本文涉及的题目一共十六道，全部都可以在 [typescript-exercises](https://typescript-exercises.github.io/) 上在线提交。



可以和标准答案进行对比。 

并且由于使用了浏览器缓存， 因此无需登录的情况下也可以保证关掉页面，你的答题进度也会保留。

想重置进度，清空缓存，无痕模式或者换浏览器都可以。

题目中涉及到的知识点我基本也都在之前的文章中提到了，如果你没有看过，强烈建议先完成前面的教程，然后将上面的题目自己做一遍之后再看本文。另外一定要按照顺序读， 因此前面的题目都是后面的铺垫。

为了不让文章太过于冗长， 本篇文章分两次发布， 一次 8 道题，一共十五道。每道题都有思路，前置知识以及代码。 **这次给大家带来的是后 6 道**

其中有一道题需要大家有函数式编程的知识， 如果大家不知道会比较难以解释。 为了避免内容太过分散，将这道题从我的题解中移除，故只有 6 道。

## 题目九

### 题目描述

Intro:  
  
 PowerUsers idea was bad. Once those users got  
 extended permissions, they started bullying others  
 and we lost a lot of great users.  
 As a response we spent all the remaining money  
 on the marketing and got even more users.  
 We need to start preparing to move everything to a  
 real database. For now we just do some mocks.  
  
 The server API format was decided to be the following:  
  
 In case of success: { status: 'success', data: RESPONSE\_DATA }  
 In case of error: { status: 'error', error: ERROR\_MESSAGE }  
  
 The API engineer started creating types for this API and  
 quickly figured out that the amount of types needed to be  
 created is too big.  
  
Exercise:  
  
 Remove UsersApiResponse and AdminsApiResponse types  
 and use generic type ApiResponse in order to specify API  
 response formats for each of the functions.

题目的大概意思是：之前都是写死的数据， 现在数据需要从接口拿，请你定义这个接口的类型。

### 题目内置代码

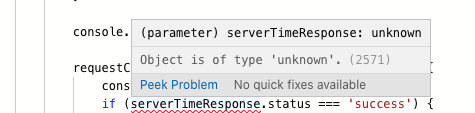
interface User {  
 type: "user";  
 name: string;  
 age: number;  
 occupation: string;  
}  
  
interface Admin {  
 type: "admin";  
 name: string;  
 age: number;  
 role: string;  
}  
  
type Person = User | Admin;  
  
const admins: Admin[] = [  
 { type: "admin", name: "Jane Doe", age: 32, role: "Administrator" },  
 { type: "admin", name: "Bruce Willis", age: 64, role: "World saver" },  
];  
  
const users: User[] = [  
 {  
 type: "user",  
 name: "Max Mustermann",  
 age: 25,  
 occupation: "Chimney sweep",  
 },  
 { type: "user", name: "Kate Müller", age: 23, occupation: "Astronaut" },  
];  
  
export type ApiResponse<T> = unknown;  
  
type AdminsApiResponse =  
 | {  
 status: "success";  
 data: Admin[];  
 }  
 | {  
 status: "error";  
 error: string;  
 };  
  
export function requestAdmins(callback: (response: AdminsApiResponse) => void) {  
 callback({  
 status: "success",  
 data: admins,  
 });  
}  
  
type UsersApiResponse =  
 | {  
 status: "success";  
 data: User[];  
 }  
 | {  
 status: "error";  
 error: string;  
 };  
  
export function requestUsers(callback: (response: UsersApiResponse) => void) {  
 callback({  
 status: "success",  
 data: users,  
 });  
}  
  
export function requestCurrentServerTime(  
 callback: (response: unknown) => void  
) {  
 callback({  
 status: "success",  
 data: Date.now(),  
 });  
}  
  
export function requestCoffeeMachineQueueLength(  
 callback: (response: unknown) => void  
) {  
 callback({  
 status: "error",  
 error: "Numeric value has exceeded Number.MAX\_SAFE\_INTEGER.",  
 });  
}  
  
function logPerson(person: Person) {  
 console.log(  
 ` - ${person.name}, ${person.age}, ${  
 person.type === "admin" ? person.role : person.occupation  
 }`  
 );  
}  
  
function startTheApp(callback: (error: Error | null) => void) {  
 requestAdmins((adminsResponse) => {  
 console.log("Admins:");  
 if (adminsResponse.status === "success") {  
 adminsResponse.data.forEach(logPerson);  
 } else {  
 return callback(new Error(adminsResponse.error));  
 }  
  
 console.log();  
  
 requestUsers((usersResponse) => {  
 console.log("Users:");  
 if (usersResponse.status === "success") {  
 usersResponse.data.forEach(logPerson);  
 } else {  
 return callback(new Error(usersResponse.error));  
 }  
  
 console.log();  
  
 requestCurrentServerTime((serverTimeResponse) => {  
 console.log("Server time:");  
 if (serverTimeResponse.status === "success") {  
 console.log(  
 ` ${new Date(serverTimeResponse.data).toLocaleString()}`  
 );  
 } else {  
 return callback(new Error(serverTimeResponse.error));  
 }  
  
 console.log();  
  
 requestCoffeeMachineQueueLength((coffeeMachineQueueLengthResponse) => {  
 console.log("Coffee machine queue length:");  
 if (coffeeMachineQueueLengthResponse.status === "success") {  
 console.log(` ${coffeeMachineQueueLengthResponse.data}`);  
 } else {  
 return callback(new Error(coffeeMachineQueueLengthResponse.error));  
 }  
  
 callback(null);  
 });  
 });  
 });  
 });  
}  
  
startTheApp((e: Error | null) => {  
 console.log();  
 if (e) {  
 console.log(  
 `Error: "${e.message}", but it's fine, sometimes errors are inevitable.`  
 );  
 } else {  
 console.log("Success!");  
 }  
});

### 前置知识

* 泛型
* 回调函数

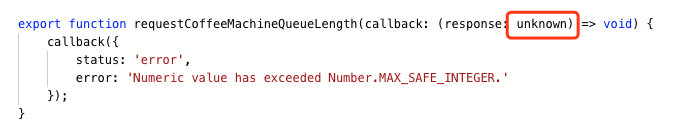
### 思路

我们还是直接看报错。



很明显这个报错的原因是类型是 unknown， 因此我们只有将 unknown 改成正确的类型即可。

换句话说， 就是把这种地方改成正确类型即可。



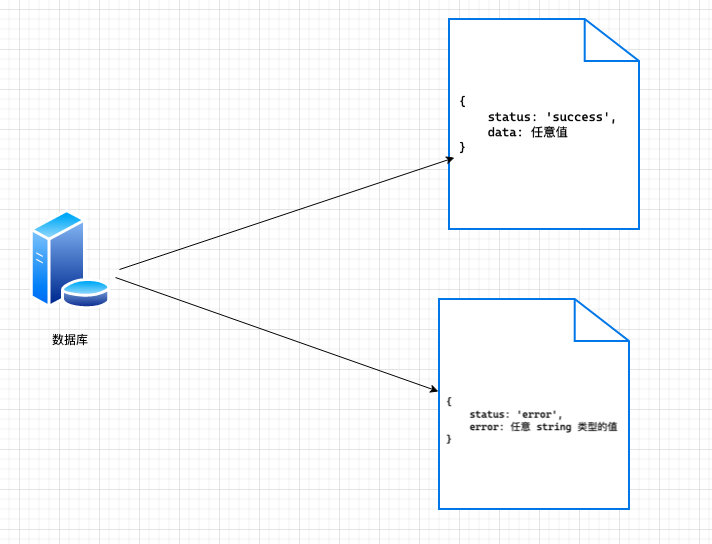
题目描述说了， 这个 response 其实是从后端返回的。 而后端返回的数据有固定的格式。比如获取用户列表接口：

type UsersApiResponse =  
 | {  
 status: "success";  
 data: User[];  
 }  
 | {  
 status: "error";  
 error: string;  
 };

其他接口也是类似， 不同的是 data 的类型。因此我们考虑使用泛型封装，将 data 的类型作为参数即可。

从本质上来说， 就是从后端取的数据有两种大的可能， 一种是错误， 一种是成功。两者在同一接口同一时刻只会出现一个，且必须出现一个。

而成功的情况又会随着接口不同从而可能产生不同的类型。



这是明显的使用 **或逻辑关系** 和**泛型进行类型定义**的强烈信号。 我们可以使用泛型做如下改造：

export type ApiResponse<T> =  
 | {  
 status: "success";  
 data: T;  
 }  
 | {  
 status: "error";  
 error: string;  
 };

那么上面的 UsersApiResponse 就可以变成：

type UsersApiResponse = ApiResponse<User[]>;

不懂的同学建议看下我之前的文章：- [你不知道的 TypeScript 泛型（万字长文，建议收藏）](https://lucifer.ren/blog/2020/06/16/ts-generics/)

用同样的套路把其他后端返回加上类型即可。

### 代码

核心代码：

export type ApiResponse<T> =  
 | {  
 status: "success";  
 data: T;  
 }  
 | {  
 status: "error";  
 error: string;  
 };  
  
export function requestAdmins(  
 callback: (response: ApiResponse<Admin[]>) => void  
) {  
 callback({  
 status: "success",  
 data: admins,  
 });  
}  
  
export function requestUsers(  
 callback: (response: ApiResponse<User[]>) => void  
) {  
 callback({  
 status: "success",  
 data: users,  
 });  
}  
  
export function requestCurrentServerTime(  
 callback: (response: ApiResponse<number>) => void  
) {  
 callback({  
 status: "success",  
 data: Date.now(),  
 });  
}  
  
export function requestCoffeeMachineQueueLength(  
 callback: (response: ApiResponse<number>) => void  
) {  
 callback({  
 status: "error",  
 error: "Numeric value has exceeded Number.MAX\_SAFE\_INTEGER.",  
 });  
}

## 题目十

### 题目描述

Intro:  
  
 We have asynchronous functions now, advanced technology.  
 This makes us a tech startup officially now.  
 But one of the consultants spoiled our dreams about  
 inevitable future IT leadership.  
 He said that callback-based asynchronicity is not  
 popular anymore and everyone should use Promises.  
 He promised that if we switch to Promises, this would  
 bring promising results.  
  
Exercise:  
  
 We don't want to reimplement all the data-requesting  
 functions. Let's decorate the old callback-based  
 functions with the new Promise-compatible result.  
 The final function should return a Promise which  
 would resolve with the final data directly  
 (i.e. users or admins) or would reject with an error  
 (or type Error).  
  
 The function should be named promisify.  
  
Higher difficulty bonus exercise:  
  
 Create a function promisifyAll which accepts an object  
 with functions and returns a new object where each of  
 the function is promisified.  
  
 Rewrite api creation accordingly:  
  
 const api = promisifyAll(oldApi);

题目大意是：前面用的是基于 callback 形式的代码， 他们对代码进行了重构，改造成了 Promise，让你对基于 Promise 的接口进行类型定义。

### 题目内置代码

interface User {  
 type: "user";  
 name: string;  
 age: number;  
 occupation: string;  
}  
  
interface Admin {  
 type: "admin";  
 name: string;  
 age: number;  
 role: string;  
}  
  
type Person = User | Admin;  
  
const admins: Admin[] = [  
 { type: "admin", name: "Jane Doe", age: 32, role: "Administrator" },  
 { type: "admin", name: "Bruce Willis", age: 64, role: "World saver" },  
];  
  
const users: User[] = [  
 {  
 type: "user",  
 name: "Max Mustermann",  
 age: 25,  
 occupation: "Chimney sweep",  
 },  
 { type: "user", name: "Kate Müller", age: 23, occupation: "Astronaut" },  
];  
  
export type ApiResponse<T> =  
 | {  
 status: "success";  
 data: T;  
 }  
 | {  
 status: "error";  
 error: string;  
 };  
  
export function promisify(arg: unknown): unknown {  
 return null;  
}  
  
const oldApi = {  
 requestAdmins(callback: (response: ApiResponse<Admin[]>) => void) {  
 callback({  
 status: "success",  
 data: admins,  
 });  
 },  
 requestUsers(callback: (response: ApiResponse<User[]>) => void) {  
 callback({  
 status: "success",  
 data: users,  
 });  
 },  
 requestCurrentServerTime(callback: (response: ApiResponse<number>) => void) {  
 callback({  
 status: "success",  
 data: Date.now(),  
 });  
 },  
 requestCoffeeMachineQueueLength(  
 callback: (response: ApiResponse<number>) => void  
 ) {  
 callback({  
 status: "error",  
 error: "Numeric value has exceeded Number.MAX\_SAFE\_INTEGER.",  
 });  
 },  
};  
  
export const api = {  
 requestAdmins: promisify(oldApi.requestAdmins),  
 requestUsers: promisify(oldApi.requestUsers),  
 requestCurrentServerTime: promisify(oldApi.requestCurrentServerTime),  
 requestCoffeeMachineQueueLength: promisify(  
 oldApi.requestCoffeeMachineQueueLength  
 ),  
};  
  
function logPerson(person: Person) {  
 console.log(  
 ` - ${person.name}, ${person.age}, ${  
 person.type === "admin" ? person.role : person.occupation  
 }`  
 );  
}  
  
async function startTheApp() {  
 console.log("Admins:");  
 (await api.requestAdmins()).forEach(logPerson);  
 console.log();  
  
 console.log("Users:");  
 (await api.requestUsers()).forEach(logPerson);  
 console.log();  
  
 console.log("Server time:");  
 console.log(  
 ` ${new Date(await api.requestCurrentServerTime()).toLocaleString()}`  
 );  
 console.log();  
  
 console.log("Coffee machine queue length:");  
 console.log(` ${await api.requestCoffeeMachineQueueLength()}`);  
}  
  
startTheApp().then(  
 () => {  
 console.log("Success!");  
 },  
 (e: Error) => {  
 console.log(  
 `Error: "${e.message}", but it's fine, sometimes errors are inevitable.`  
 );  
 }  
);

### 前置知识

* Promise
* promisify
* 泛型
* 高阶函数

### 思路

题目给了一个 promisefy， 并且类型都是 unknown，不难看出， 它就是想让我们改造 promisefy 使其不报错， 并能正确推导类型。

export function promisify(arg: unknown): unknown {  
 return null;  
}

我们先不考虑这个类型怎么写，先把 promiify 实现一下再说。这需要你有一点高阶函数和 promise 的知识。由于这不是本文的重点，因此不赘述。

export function promisify(fn) {  
 return () =>  
 new Promise((resolve, reject) => {  
 fn((response) => {  
 if (response.status === "success") resolve(response.data);  
 else reject(response.error);  
 });  
 });  
}

接下来，我们需要给其增加类型签名。

这个 fn 实际上是一个函数，并且又接受一个 callback 作为参数。 因此大概是这个样子：

((something) = void) => void

这里的 something 实际上我们在上一节已经解决了，直接套用即可。代码：

(callback: (response: ApiResponse<T>) => void) => void

整体代码大概是：

export function promisify<T>(  
 fn: (callback: (response: ApiResponse<T>) => void) => void  
): () => Promise<T> {  
 // 上面的实现  
}

### 代码

核心代码：

export function promisify<T>(  
 fn: (callback: (response: ApiResponse<T>) => void) => void  
): () => Promise<T> {  
 return () =>  
 new Promise((resolve, reject) => {  
 fn((response) => {  
 if (response.status === "success") resolve(response.data);  
 else reject(response.error);  
 });  
 });  
}

## 第十一题

### 题目描述

Intro:  
  
 In order to engage users in the communication with  
 each other we have decided to decorate usernames  
 in various ways. A brief search led us to a library  
 called "str-utils". Bad thing is that it lacks  
 TypeScript declarations.  
  
Exercise:  
  
 Check str-utils module implementation at:  
 node\_modules/str-utils/index.js  
 node\_modules/str-utils/README.md  
  
 Provide type declaration for that module in:  
 declarations/str-utils/index.d.ts  
  
 Try to avoid duplicates of type declarations,  
 use type aliases.

题目的意思是他们用到了一个库 str-utils，这个库的人又没给我们写类型定义，于是我们不得不去自己写（好真实的例子啊）。

其实就是让我们实现以下函数的类型签名：

import {  
 strReverse,  
 strToLower,  
 strToUpper,  
 strRandomize,  
 strInvertCase,  
} from "str-utils";

### 题目内置代码

// declarations/str-utils/index.d.js  
declare module "str-utils" {  
 // export const ...  
 // export function ...  
}  
  
// index.ts  
import {  
 strReverse,  
 strToLower,  
 strToUpper,  
 strRandomize,  
 strInvertCase,  
} from "str-utils";  
  
interface User {  
 type: "user";  
 name: string;  
 age: number;  
 occupation: string;  
}  
  
interface Admin {  
 type: "admin";  
 name: string;  
 age: number;  
 role: string;  
}  
  
type Person = User | Admin;  
  
const admins: Admin[] = [  
 { type: "admin", name: "Jane Doe", age: 32, role: "Administrator" },  
 { type: "admin", name: "Bruce Willis", age: 64, role: "World saver" },  
 { type: "admin", name: "Steve", age: 40, role: "Steve" },  
 { type: "admin", name: "Will Bruces", age: 30, role: "Overseer" },  
 { type: "admin", name: "Superwoman", age: 28, role: "Customer support" },  
];  
  
const users: User[] = [  
 {  
 type: "user",  
 name: "Max Mustermann",  
 age: 25,  
 occupation: "Chimney sweep",  
 },  
 { type: "user", name: "Kate Müller", age: 23, occupation: "Astronaut" },  
 { type: "user", name: "Moses", age: 70, occupation: "Desert guide" },  
 { type: "user", name: "Superman", age: 28, occupation: "Ordinary person" },  
 { type: "user", name: "Inspector Gadget", age: 31, occupation: "Undercover" },  
];  
  
const isAdmin = (person: Person): person is Admin => person.type === "admin";  
const isUser = (person: Person): person is User => person.type === "user";  
  
export const nameDecorators = [  
 strReverse,  
 strToLower,  
 strToUpper,  
 strRandomize,  
 strInvertCase,  
];  
  
function logPerson(person: Person) {  
 let additionalInformation: string = "";  
 if (isAdmin(person)) {  
 additionalInformation = person.role;  
 }  
 if (isUser(person)) {  
 additionalInformation = person.occupation;  
 }  
 const randomNameDecorator =  
 nameDecorators[Math.round(Math.random() \* (nameDecorators.length - 1))];  
 const name = randomNameDecorator(person.name);  
 console.log(` - ${name}, ${person.age}, ${additionalInformation}`);  
}  
  
([] as Person[]).concat(users, admins).forEach(logPerson);  
  
// In case if you are stuck:  
// https://www.typescriptlang.org/docs/handbook/modules.html#ambient-modules

### 前置知识

* 如何给缺乏类型定义的第三方库定义类型

### 思路

这个题目的考点就是**如何给缺乏类型定义的第三方库定义类型**。

这个时候我们只要新建一个文件然后加入以下代码即可。

declare module "str-utils" {  
 // 在这里定义类型  
 // export const ...  
 // export function ...  
}

其中 str-utils 是那个可恶的没有类型定义的库的名字。

有了这个知识，我们的代码就简单了。

### 代码

declare module "str-utils" {  
 // export const ...  
 // export function ...  
 export function strReverse(s: string): string;  
 export function strToLower(s: string): string;  
 export function strToUpper(s: string): string;  
 export function strRandomize(s: string): string;  
 export function strInvertCase(s: string): string;  
}

## 第十二题

### 题目描述

Intro:  
  
 We have so many users and admins in the database!  
 CEO's father Jeff says that we are a BigData  
 startup now. We have no idea what it means, but  
 Jeff says that we need to do some statistics and  
 analytics.  
  
 We've ran a questionnaire within the team to  
 figure out what do we know about statistics.  
 The only person who filled it was our coffee  
 machine maintainer. The answers were:  
  
 \* Maximums  
 \* Minumums  
 \* Medians  
 \* Averages  
  
 We found a piece of code on stackoverflow and  
 compiled it into a module `stats`. The bad  
 thing is that it lacks type declarations.  
  
Exercise:  
  
 Check stats module implementation at:  
 node\_modules/stats/index.js  
 node\_modules/stats/README.md  
  
 Provide type declaration for that module in:  
 declarations/stats/index.d.ts  
  
Higher difficulty bonus exercise:  
  
 Avoid duplicates of type declarations.

题目大概意思是又来了一个库，这个库又没有写定义，我们又要自己写。 （真实++）

### 题目内置代码

// declartions/stats/index.d.ts  
declare module "stats" {  
 export function getMaxIndex(input: unknown, comparator: unknown): unknown;  
}  
  
// index.ts  
import {  
 getMaxIndex,  
 getMaxElement,  
 getMinIndex,  
 getMinElement,  
 getMedianIndex,  
 getMedianElement,  
 getAverageValue,  
} from "stats";  
  
interface User {  
 type: "user";  
 name: string;  
 age: number;  
 occupation: string;  
}  
  
interface Admin {  
 type: "admin";  
 name: string;  
 age: number;  
 role: string;  
}  
  
const admins: Admin[] = [  
 { type: "admin", name: "Jane Doe", age: 32, role: "Administrator" },  
 { type: "admin", name: "Bruce Willis", age: 64, role: "World saver" },  
 { type: "admin", name: "Steve", age: 40, role: "Steve" },  
 { type: "admin", name: "Will Bruces", age: 30, role: "Overseer" },  
 { type: "admin", name: "Superwoman", age: 28, role: "Customer support" },  
];  
  
const users: User[] = [  
 {  
 type: "user",  
 name: "Max Mustermann",  
 age: 25,  
 occupation: "Chimney sweep",  
 },  
 { type: "user", name: "Kate Müller", age: 23, occupation: "Astronaut" },  
 { type: "user", name: "Moses", age: 70, occupation: "Desert guide" },  
 { type: "user", name: "Superman", age: 28, occupation: "Ordinary person" },  
 { type: "user", name: "Inspector Gadget", age: 31, occupation: "Undercover" },  
];  
  
function logUser(user: User | null) {  
 if (!user) {  
 console.log(" - none");  
 return;  
 }  
 const pos = users.indexOf(user) + 1;  
 console.log(` - #${pos} User: ${user.name}, ${user.age}, ${user.occupation}`);  
}  
  
function logAdmin(admin: Admin | null) {  
 if (!admin) {  
 console.log(" - none");  
 return;  
 }  
 const pos = admins.indexOf(admin) + 1;  
 console.log(` - #${pos} Admin: ${admin.name}, ${admin.age}, ${admin.role}`);  
}  
  
const compareUsers = (a: User, b: User) => a.age - b.age;  
const compareAdmins = (a: Admin, b: Admin) => a.age - b.age;  
const colorizeIndex = (value: number) => String(value + 1);  
  
export {  
 getMaxIndex,  
 getMaxElement,  
 getMinIndex,  
 getMinElement,  
 getMedianIndex,  
 getMedianElement,  
 getAverageValue,  
};  
  
console.log("Youngest user:");  
logUser(getMinElement(users, compareUsers));  
console.log(  
 ` - was ${colorizeIndex(getMinIndex(users, compareUsers))}th to register`  
);  
  
console.log();  
  
console.log("Median user:");  
logUser(getMedianElement(users, compareUsers));  
console.log(  
 ` - was ${colorizeIndex(getMedianIndex(users, compareUsers))}th to register`  
);  
  
console.log();  
  
console.log("Oldest user:");  
logUser(getMaxElement(users, compareUsers));  
console.log(  
 ` - was ${colorizeIndex(getMaxIndex(users, compareUsers))}th to register`  
);  
  
console.log();  
  
console.log("Average user age:");  
console.log(  
 ` - ${String(getAverageValue(users, ({ age }: User) => age))} years`  
);  
  
console.log();  
  
console.log("Youngest admin:");  
logAdmin(getMinElement(admins, compareAdmins));  
console.log(  
 ` - was ${colorizeIndex(getMinIndex(users, compareUsers))}th to register`  
);  
  
console.log();  
  
console.log("Median admin:");  
logAdmin(getMedianElement(admins, compareAdmins));  
console.log(  
 ` - was ${colorizeIndex(getMedianIndex(users, compareUsers))}th to register`  
);  
  
console.log();  
  
console.log("Oldest admin:");  
logAdmin(getMaxElement(admins, compareAdmins));  
console.log(  
 ` - was ${colorizeIndex(getMaxIndex(users, compareUsers))}th to register`  
);  
  
console.log();  
  
console.log("Average admin age:");  
console.log(  
 ` - ${String(getAverageValue(admins, ({ age }: Admin) => age))} years`  
);

### 前置知识

* 泛型
* 高阶函数
* 如何给缺乏类型定义的第三方库定义类型

### 思路

和上面的思路类似。 唯一的不同的是这道题的需要实现的几个方法支持不同的入参类型。

import {  
 getMaxIndex,  
 getMaxElement,  
 getMinIndex,  
 getMinElement,  
 getMedianIndex,  
 getMedianElement,  
 getAverageValue,  
} from "stats";

因此，我们考虑使用泛型来定义。 知道了这个， 代码就不难写。 这是最最基本的泛型， 比我们前面写的还简单。

### 代码

declare module "stats" {  
 export function getMaxIndex<T>(  
 input: T[],  
 comparator: (a: T, b: T) => number  
 ): number;  
 export function getMaxElement<T>(  
 input: T[],  
 comparator: (a: T, b: T) => number  
 ): T;  
 export function getMinElement<T>(  
 input: T[],  
 comparator: (a: T, b: T) => number  
 ): T;  
 export function getMedianIndex<T>(  
 input: T[],  
 comparator: (a: T, b: T) => number  
 ): number;  
 export function getMedianElement<T>(  
 input: T[],  
 comparator: (a: T, b: T) => number  
 ): T;  
 export function getAverageValue<T>(  
 input: T[],  
 getValue: (a: T) => number  
 ): number;  
 export function getMinIndex<T>(  
 input: T[],  
 comparator: (a: T, b: T) => number  
 ): number;  
}

## 第十三题

### 题目描述

Intro:  
  
 The next logical step for us is to provide more  
 precise registration date for our users and admins.  
 We've approximately made up dates for each user and  
 admin and used a library called "date-wizard" in  
 order to pretty-format the dates.  
  
 Unfortunately, type declarations which came with  
 "date-wizard" library were incomplete.  
  
 1. DateDetails interface is missing  
 time related fields such as hours, minutes and  
 seconds.  
 2. Function "pad" is exported but not declared.  
  
Exercise:  
  
 Check date-wizard module implementation at:  
 node\_modules/date-wizard/index.js  
 node\_modules/date-wizard/index.d.ts  
  
 Extend type declaration of that module in:  
 module-augmentations/date-wizard/index.ts

题目大概意思是又来了一个库，这个库又没有写定义，我们又要自己写。 （真实+++++++++++++）

### 题目内置代码

// module-augmentations/data-wizard/index.d.ts  
  
// This enables module augmentation mode.  
import "date-wizard";  
  
declare module "date-wizard" {  
 // Add your module extensions here.  
}  
  
// index.ts  
import \* as dateWizard from "date-wizard";  
import "./module-augmentations/date-wizard";  
  
interface User {  
 type: "user";  
 name: string;  
 age: number;  
 occupation: string;  
 registered: Date;  
}  
  
interface Admin {  
 type: "admin";  
 name: string;  
 age: number;  
 role: string;  
 registered: Date;  
}  
  
type Person = User | Admin;  
  
const admins: Admin[] = [  
 {  
 type: "admin",  
 name: "Jane Doe",  
 age: 32,  
 role: "Administrator",  
 registered: new Date("2016-06-01T16:23:13"),  
 },  
 {  
 type: "admin",  
 name: "Bruce Willis",  
 age: 64,  
 role: "World saver",  
 registered: new Date("2017-02-11T12:12:11"),  
 },  
 {  
 type: "admin",  
 name: "Steve",  
 age: 40,  
 role: "Steve",  
 registered: new Date("2018-01-05T11:02:30"),  
 },  
 {  
 type: "admin",  
 name: "Will Bruces",  
 age: 30,  
 role: "Overseer",  
 registered: new Date("2018-08-12T10:01:24"),  
 },  
 {  
 type: "admin",  
 name: "Superwoman",  
 age: 28,  
 role: "Customer support",  
 registered: new Date("2019-03-25T07:51:05"),  
 },  
];  
  
const users: User[] = [  
 {  
 type: "user",  
 name: "Max Mustermann",  
 age: 25,  
 occupation: "Chimney sweep",  
 registered: new Date("2016-02-15T09:25:13"),  
 },  
 {  
 type: "user",  
 name: "Kate Müller",  
 age: 23,  
 occupation: "Astronaut",  
 registered: new Date("2016-03-23T12:47:03"),  
 },  
 {  
 type: "user",  
 name: "Moses",  
 age: 70,  
 occupation: "Desert guide",  
 registered: new Date("2017-02-19T17:22:56"),  
 },  
 {  
 type: "user",  
 name: "Superman",  
 age: 28,  
 occupation: "Ordinary person",  
 registered: new Date("2018-02-25T19:44:28"),  
 },  
 {  
 type: "user",  
 name: "Inspector Gadget",  
 age: 31,  
 occupation: "Undercover",  
 registered: new Date("2019-03-25T09:29:12"),  
 },  
];  
  
const isAdmin = (person: Person): person is Admin => person.type === "admin";  
const isUser = (person: Person): person is User => person.type === "user";  
  
function logPerson(person: Person, index: number) {  
 let additionalInformation: string = "";  
 if (isAdmin(person)) {  
 additionalInformation = person.role;  
 }  
 if (isUser(person)) {  
 additionalInformation = person.occupation;  
 }  
 let registeredAt = dateWizard(  
 person.registered,  
 "{date}.{month}.{year} {hours}:{minutes}"  
 );  
 let num = `#${dateWizard.pad(index + 1)}`;  
 console.log(  
 ` - ${num}: ${person.name}, ${person.age}, ${additionalInformation}, ${registeredAt}`  
 );  
}  
  
export { dateWizard };  
  
console.log("All users:");  
  
([] as Person[]).concat(users, admins).forEach(logPerson);  
  
console.log();  
  
console.log("Early birds:");  
  
([] as Person[])  
 .concat(users, admins)  
 .filter((person) => dateWizard.dateDetails(person.registered).hours < 10)  
 .forEach(logPerson);  
  
// In case if you are stuck:  
// https://www.typescriptlang.org/docs/handbook/modules.html#ambient-modules  
// https://www.typescriptlang.org/docs/handbook/declaration-merging.html

### 前置知识

* interface 或 type 声明自定义类型
* 如何给缺乏类型定义的第三方库定义类型

### 思路

和上面两道题思路一样， 不用多说了吧？

### 代码

// This enables module augmentation mode.  
import "date-wizard";  
  
declare module "date-wizard" {  
 // Add your module extensions here.  
 function dateWizard(date: string, format: string): string;  
 function pad(s: number): string;  
 interface DateDetails {  
 year: number;  
 month: number;  
 date: number;  
 hours: number;  
 minutes: number;  
 seconds: number;  
 }  
 function dateDetails(date: Date): DateDetails;  
}

## 第十四题

需要大家有函数式编程的知识， 如果大家不知道会比较难以解释。 为了避免内容太过分散，将这道题从我的题解中移除。

对函数式编程感兴趣的，也可是看下我之前写的文章 [函数式编程系列教程](https://github.com/azl397985856/functional-programming)。

## 第十五题

### 题目描述

Intro:  
  
 Our attempt to Open Source didn't work quite as  
 expected. It turned out there were already many  
 existing functional JS libraries.  
  
 All the remaining developers left the company as  
 well. It seems that they are joining a very  
 ambitious startup which re-invented a juicer and  
 raised millions of dollars.  
 Too bad we cannot compete with this kind of  
 financing even though we believe our idea is  
 great.  
  
 It's time to shine for the last time and publish  
 our new invention: object-constructor as our CTO  
 named it. A small library which helps  
 manipulating an object.  
  
Exercise:  
  
 Here is a library which helps manipulating objects.  
 We tried to write type annotations and we failed.  
 Please help!

题目大概意思是函数式编程他们 hold 不住，于是又准备切换到面向对象编程。 于是你需要补充类型定义使得代码不报错。

### 题目内置代码

export class ObjectManipulator {  
 constructor(protected obj) {}  
  
 public set(key, value) {  
 return new ObjectManipulator({ ...this.obj, [key]: value });  
 }  
  
 public get(key) {  
 return this.obj[key];  
 }  
  
 public delete(key) {  
 const newObj = { ...this.obj };  
 delete newObj[key];  
 return new ObjectManipulator(newObj);  
 }  
  
 public getObject() {  
 return this.obj;  
 }  
}

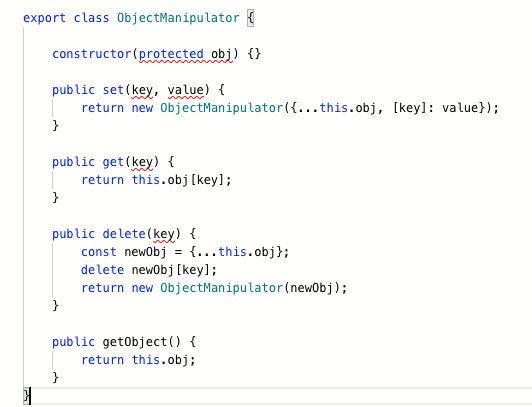
### 前置知识

* 泛型
* Omit 泛型
* ES6 class
* keyof
* 使用 extends 进行泛型约束
* 联合类型

### 思路

这道题难度颇高，比前面的泛型题目都要难。 也是本系列的压轴题，我们重点讲一下。

首先题目有五个报错位置， 报错信息都是隐式使用了 any ， 因此我们的思路就是将五个地方显式声明类型即可。



从它的名字 ObjectManipulator 以及 api 可以看出， 它应该可以存储任何对象，因此使用泛型定义就不难想到。

你也可是把这个 ObjectManipulator 想象成抽象包包。 你的期望是限量款包包拍照的时候用，普通包包和闺蜜逛街的时候用，优衣库送的包包逛超市的时候用等等。

ObjectManipulator 是一个抽象的包包概念，不是具体的包， 比如当你买一个 LV 的包包的时候就是 ObjectManipulator<LVBag>。这样当你往 LV 里放超市买的水果的时候就可以报错：你怎么可以用 LV 包包装这样东西呢？你应该用 ta 装\*。



当然这个例子很不严谨， 这个只是帮助大家快速理解而已，切莫较真。

理解了题意，我们就可以开始写了。

我们先改第一个错 - 构造函数 constructor， 这个错比较简单。

export class ObjectManipulator<T> {  
 constructor(protected obj: T) {  
 this.obj = obj;  
 }  
 ...  
}

这个时候经过 ObjectManipulator 实例化产生的对象的 this.obj 都是 T 类型，其中 T 是泛型。因此 getObject 的错也不难改，返回值写 T 就行。

export class ObjectManipulator<T> {  
 ...  
 public getObject(): T {  
 return this.obj;  
 }  
}

剩下的 get，set 和 delete 思路有点类似。 先拿 get 来说：

export class ObjectManipulator<T> {  
 ...  
 public get(key) {  
 return this.obj[key];  
 }  
 ...  
}

这个怎么写类型呢？ key 理论上可是是任何值，返回值理论上也可以是任何值。但是一旦类型 T 确定了， 那么实际上 key 和返回值就不是任意值了。 比如：

type A = ObjectManipulator<{ name: string; age: number }>;  
const a: A = new ObjectManipulator({ name: "", age: 17 });

如上代码中的 A 是 ObjectManipulator 传入具体类型 { name: string; age: number } 产生的新的类型。

我这里用的是行内类型， 实际项目建议使用 interface 或者 type 定义类型。

之后我们模拟一些操作：

a.set("name", "脑洞前端");  
a.get("name");  
a.get("name123"); // 期望报错  
a.set("name123", "脑洞");  
a.delete("name123"); // 期望报错  
a.delete("name");

实际上，我**可能**期望的是其中一些行为可以借助 TypeScript 的类型分析直接报错。

简单来说，我的期望是 **get 和 delete 不在 T 中的 key 都报错。**

当然你的真实项目也可以不认同我的观点， 比如 get 一个不在 T 中定义的 key 也可以，但是我还是推荐你这么做。

知道了这个， 再结合我之前有关泛型的文章就不难写出来。

其中 get 和 delete 的代码：

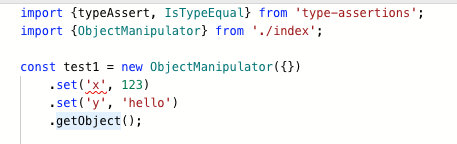
export class ObjectManipulator<T> {  
 public get<K extends keyof T>(key: K): T[K] {  
 return this.obj[key];  
 }  
  
 public delete<K extends keyof T>(key: K): ObjectManipulator<Omit<T, K>> {  
 const newObj = { ...this.obj };  
 delete newObj[key];  
 return new ObjectManipulator(newObj);  
 }  
}

最后是 set，其实一开始我的 set 是这么写的。

export class ObjectManipulator<T> {  
 public set<K extends keyof T, V>(key: K, value: V): ObjectManipulator<T> {  
 return new ObjectManipulator({  
 ...this.obj,  
 [key]: value,  
 }) as ObjectManipulator<T & { [k in K]: V }>;  
 }  
}

但是无奈没有通过官方的测试用例。 实际项目我其实更推荐我上面的这种写法。下面是我为了通过所有的测试用例写的方法。

经过分析， 我发现它期望的是 set 中的 key 可以不是 T 中的。这一点从官方给的测试用例就可以看出来。



因此我将代码改成 K 放宽到任意 string，返回值做了一个联合类型。代码：

export class ObjectManipulator<T> {  
 ...  
 public set<K extends string, V>(  
 key: K,  
 value: V  
 ): ObjectManipulator<T & { [k in K]: V }> {  
 return new ObjectManipulator({  
 ...this.obj,  
 [key]: value,  
 }) as ObjectManipulator<T & { [k in K]: V }>;  
 }  
 ...  
}

终于通过了所有的测试用例。

### 代码

export class ObjectManipulator<T> {  
 constructor(protected obj: T) {  
 this.obj = obj;  
 }  
 public set<K extends string, V>(  
 key: K,  
 value: V  
 ): ObjectManipulator<T & { [k in K]: V }> {  
 return new ObjectManipulator({  
 ...this.obj,  
 [key]: value,  
 }) as ObjectManipulator<T & { [k in K]: V }>;  
 }  
  
 public get<K extends keyof T>(key: K): T[K] {  
 return this.obj[key];  
 }  
  
 public delete<K extends keyof T>(key: K): ObjectManipulator<Omit<T, K>> {  
 const newObj = { ...this.obj };  
 delete newObj[key];  
 return new ObjectManipulator(newObj);  
 }  
  
 public getObject(): T {  
 return this.obj;  
 }  
}

## 总结

以上就是给大家带来的题目解析。 这六道题的考点有，按照我个人理解的重要程度划分为：

* type 和 interface 的基本操作（必须掌握）
* 如何给缺乏类型定义的第三方库定义类型（必须掌握）
* 联合类型 和 交叉类型（强烈建议掌握）
* 类型断言和类型收缩（强烈建议掌握）
* 泛型和常见内置泛型（强烈建议掌握）
* 高阶函数的类型定义（强烈建议掌握）

最后祝愿大家告别 anyscript，成为 TypeScript 魔法师。

## 关注我

大家也可以关注我的公众号《脑洞前端》获取更多更新鲜的前端硬核文章，带你认识你不知道的前端。



公众号【 [力扣加加](https://tva1.sinaimg.cn/large/007S8ZIlly1gfcuzagjalj30p00dwabs.jpg)】 知乎专栏【 [Lucifer - 知乎](https://www.zhihu.com/people/lu-xiao-13-70)】

点关注，不迷路！