TypeScript 练习题

2020-09-27

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（就是本文）

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。

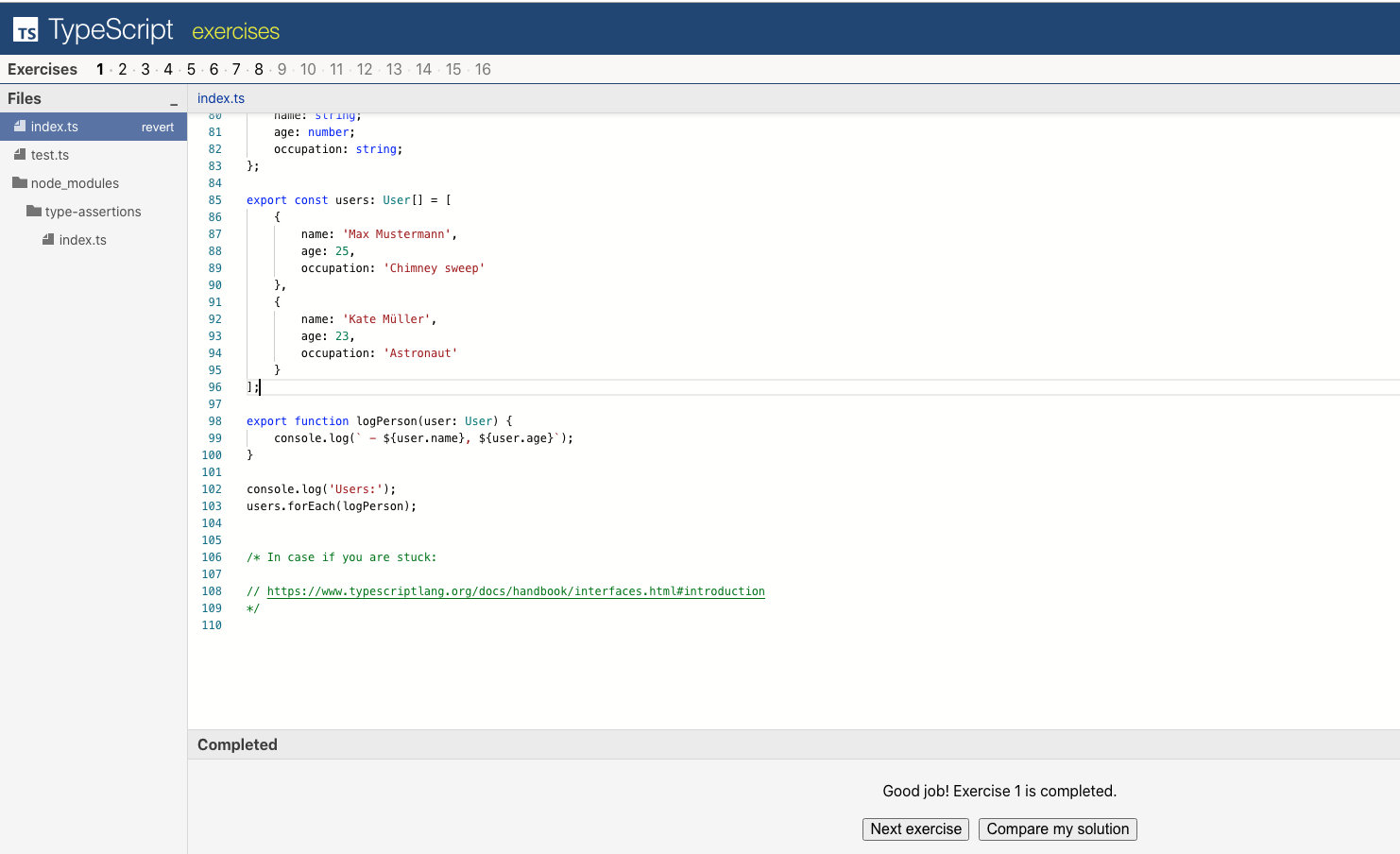
* [深入理解 TypeScript](https://jkchao.github.io/typescript-book-chinese/)
* [TypeScript 官方文档](https://www.typescriptlang.org/docs/home)

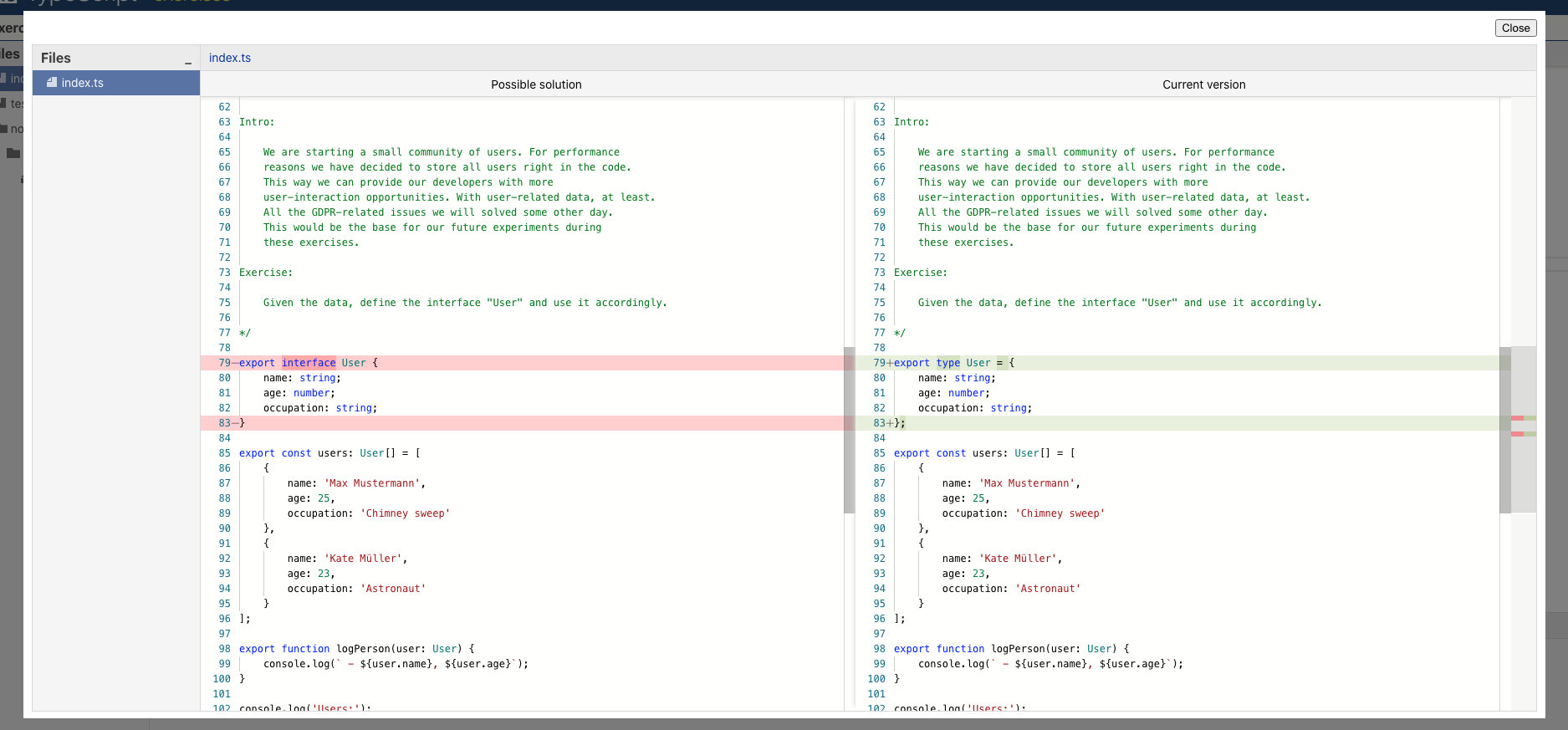
结合这两个资料和我的系列教程，掌握 TypeScript 指日可待。

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十五道，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， 这一次是 8 道题，一共十五道。每道题都有思路，前置知识以及代码。

## 题目一

### 题目描述

Intro:  
  
 We are starting a small community of users. For performance  
 reasons we have decided to store all users right in the code.  
 This way we can provide our developers with more  
 user-interaction opportunities. With user-related data, at least.  
 All the GDPR-related issues we will solved some other day.  
 This would be the base for our future experiments during  
 these exercises.  
  
Exercise:  
  
 Given the data, define the interface "User" and use it accordingly.

题目的大概意思是让你定义一个类型 User， 使得代码可以正常运行。

### 题目内置代码

export type User = unknown;  
  
export const users: unknown[] = [  
 {  
 name: "Max Mustermann",  
 age: 25,  
 occupation: "Chimney sweep",  
 },  
 {  
 name: "Kate Müller",  
 age: 23,  
 occupation: "Astronaut",  
 },  
];  
  
export function logPerson(user: unknown) {  
 console.log(` - ${user.name}, ${user.age}`);  
}  
  
console.log("Users:");  
users.forEach(logPerson);

### 前置知识

* interface 或 type 声明自定义类型

### 思路

这道题比较简单， 我们只有定义一个 User 类即可。从 users 数组中不难看出， User 中有三个属性 name ，age 和 occupation，类型分别为 string， number 和 string。因此直接使用 type 或者 interface 定义自定义类型即可。

### 代码

核心代码：

export type User = {  
 name: string;  
 age: number;  
 occupation: string;  
};

## 题目二

### 题目描述

Intro:  
  
 All 2 users liked the idea of the community. We should go  
 forward and introduce some order. We are in Germany after all.  
 Let's add a couple of admins.  
  
 Initially we only had users in the in-memory database. After  
 introducing Admins, we need to fix the types so that  
 everything works well together.  
  
Exercise:  
  
 Type "Person" is missing, please define it and use  
 it in persons array and logPerson function in order to fix  
 all the TS errors.

题目大意是补充 Person 类， 使得代码不报错。

### 题目内置代码

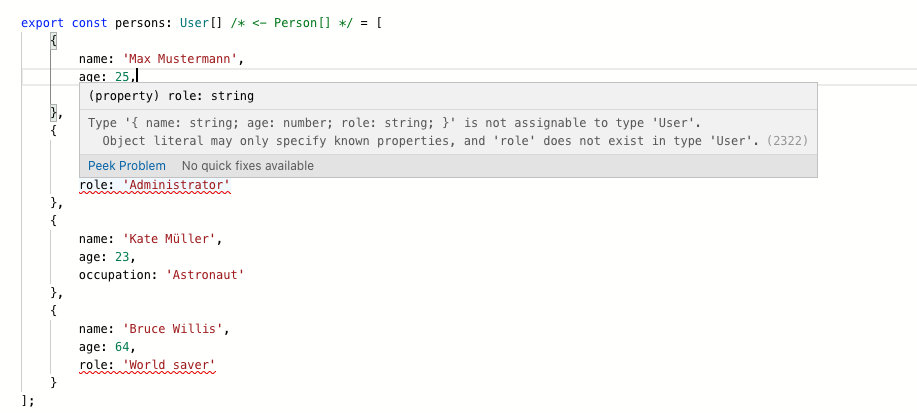
interface User {  
 name: string;  
 age: number;  
 occupation: string;  
}  
  
interface Admin {  
 name: string;  
 age: number;  
 role: string;  
}  
  
export type Person = unknown;  
  
export const persons: User[] /\* <- Person[] \*/ = [  
 {  
 name: "Max Mustermann",  
 age: 25,  
 occupation: "Chimney sweep",  
 },  
 {  
 name: "Jane Doe",  
 age: 32,  
 role: "Administrator",  
 },  
 {  
 name: "Kate Müller",  
 age: 23,  
 occupation: "Astronaut",  
 },  
 {  
 name: "Bruce Willis",  
 age: 64,  
 role: "World saver",  
 },  
];  
  
export function logPerson(user: User) {  
 console.log(` - ${user.name}, ${user.age}`);  
}  
  
persons.forEach(logPerson);

### 前置知识

* 联合类型

### 思路

我们直接从报错入手。



不难发现 persons 数组既有 User 又有 Admin。 因此 person 的函数签名应该是两者的联合类型。而题目又让我们补充 Person，于是代码将 Person 定义为 Admin 和 User 的联合类型就不难想到。

### 代码

核心代码：

export type Person = User | Admin;

这个时候， persons 数组使用的过程只能用 User 和 Admin 的共有属性， 也就是 name 和 age，这点后面的题目也会提到。 因此如果你使用了 role 或者 occupation 就会报错。怎么解决呢？ 我们继续看下一题。

## 第三题

### 题目描述

Intro:  
  
 Since we already have some of the additional  
 information about our users, it's a good idea  
 to output it in a nice way.  
  
Exercise:  
  
 Fix type errors in logPerson function.  
  
 logPerson function should accept both User and Admin  
 and should output relevant information according to  
 the input: occupation for User and role for Admin.

### 题目内置代码

interface User {  
 name: string;  
 age: number;  
 occupation: string;  
}  
  
interface Admin {  
 name: string;  
 age: number;  
 role: string;  
}  
  
export type Person = User | Admin;  
  
export const persons: Person[] = [  
 {  
 name: "Max Mustermann",  
 age: 25,  
 occupation: "Chimney sweep",  
 },  
 {  
 name: "Jane Doe",  
 age: 32,  
 role: "Administrator",  
 },  
 {  
 name: "Kate Müller",  
 age: 23,  
 occupation: "Astronaut",  
 },  
 {  
 name: "Bruce Willis",  
 age: 64,  
 role: "World saver",  
 },  
];  
  
export function logPerson(person: Person) {  
 let additionalInformation: string;  
 if (person.role) {  
 additionalInformation = person.role;  
 } else {  
 additionalInformation = person.occupation;  
 }  
 console.log(` - ${person.name}, ${person.age}, ${additionalInformation}`);  
}  
  
persons.forEach(logPerson);

### 前置知识

* 类型断言
* 类型收敛
* in 操作符

### 思路

关于类型收敛， 我在 [TypeScript 类型系统](https://lucifer.ren/blog/2020/08/15/ts-type-system/) 做了很详情的讨论。

上面代码报错的原因前面已经讲过了， 那么如何解决呢？由于 person 可能是 User ，也可能是 Admin 类型，而 TypeScript 没有足够的信息确定具体是哪一种。因此你使用 User 或者 Admin 特有的属性就会报错了。

因此解决方案的基本思想就是告诉 TypeScript **person 当前是 Admin 还是 User 类型**。有多种方式可以解决这个问题。

1. 将 person 断言为准确的类型。 就是告诉 TypeScript ”交给我吧， person 就是 xxx 类型，有错就我的锅“。

代码：

if ((<Admin>person).role) {  
 additionalInformation = (<Admin>person).role;  
} else {  
 additionalInformation = (<User>person).occupation;  
}

1. 另外一种方式是使用类型收缩，比如 is ， in， typeof ， instanceof 等。使得 Typescript 能够 Get 到当前的类型。”哦， person 上有 role 属性啊，那它就是 Admin 类型，有问题我 Typescript 的锅“

这里我们使用 in 操作符，写起来也很简单。

推荐哪种不用我多说了吧 ?

### 代码

if ("role" in person) {  
 // person 会被自动推导为 Admin  
 additionalInformation = person.role;  
} else {  
 // Person 会被自动推导为 User  
 additionalInformation = person.occupation;  
}

## 第四题

### 题目描述

Intro:  
  
 As we introduced "type" to both User and Admin  
 it's now easier to distinguish between them.  
 Once object type checking logic was extracted  
 into separate functions isUser and isAdmin -  
 logPerson function got new type errors.  
  
Exercise:  
  
 Figure out how to help TypeScript understand types in  
 this situation and apply necessary fixes.

大概意思还是让你改代码， 使得 Typescript 能理解（不报错）。

### 题目内置代码

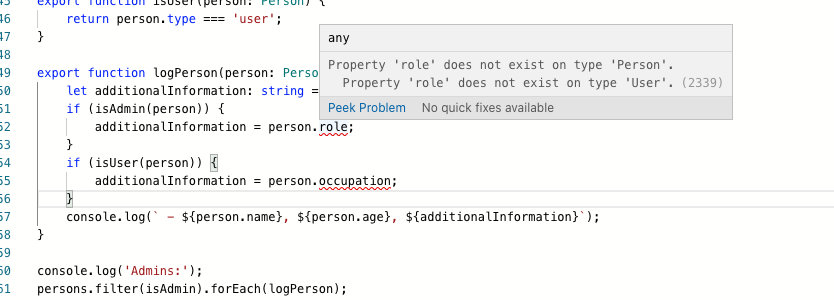
interface User {  
 type: "user";  
 name: string;  
 age: number;  
 occupation: string;  
}  
  
interface Admin {  
 type: "admin";  
 name: string;  
 age: number;  
 role: string;  
}  
  
export type Person = User | Admin;  
  
export const persons: Person[] = [  
 {  
 type: "user",  
 name: "Max Mustermann",  
 age: 25,  
 occupation: "Chimney sweep",  
 },  
 { type: "admin", name: "Jane Doe", age: 32, role: "Administrator" },  
 { type: "user", name: "Kate Müller", age: 23, occupation: "Astronaut" },  
 { type: "admin", name: "Bruce Willis", age: 64, role: "World saver" },  
];  
  
export function isAdmin(person: Person) {  
 return person.type === "admin";  
}  
  
export function isUser(person: Person) {  
 return person.type === "user";  
}  
  
export function logPerson(person: Person) {  
 let additionalInformation: string = "";  
 if (isAdmin(person)) {  
 additionalInformation = person.role;  
 }  
 if (isUser(person)) {  
 additionalInformation = person.occupation;  
 }  
 console.log(` - ${person.name}, ${person.age}, ${additionalInformation}`);  
}  
  
console.log("Admins:");  
persons.filter(isAdmin).forEach(logPerson);  
  
console.log();  
  
console.log("Users:");  
persons.filter(isUser).forEach(logPerson);

### 前置知识

* 类型收敛
* is 操作符

### 思路

我们仍然从报错入手。



实际上还是 person 的类型问题， 没有被收缩到正确的类型。看题目的代码，期望效果应该是如果进入 isAdmin 内部，那么 person 就是 Admin 类型，同理进入 isUser 内部，那么 person 就是 User 类型。

继续看下 isAdmin 和 isUser 的实现：

export function isAdmin(person: Person) {  
 return person.type === "admin";  
}  
  
export function isUser(person: Person) {  
 return person.type === "user";  
}

这里我们期望的效果是如果 isAdmin 函数返回 true ，那么 person 就应该被收敛为 Admin，isUser 同理。

这里就需要用到 is 操作符。

上文提到了类型收敛常见的操作符是 is ， in， typeof ， instanceof

### 代码

export function isAdmin(person: Person): person is Admin {  
 return person.type === "admin";  
}  
  
export function isUser(person: Person): person is User {  
 return person.type === "user";  
}

这样当 isAdmin 返回 true， 那么 person 变量就会被推导成 Admin 类型，而不是联合类型， 也就是类型发生了收缩。

不难看出，这样的类型断言会直接影响到调用 isAdmin 或 isUser 的**函数的入参的类型**。

## 第五题

### 题目描述

Intro:  
  
 Time to filter the data! In order to be flexible  
 we filter users using a number of criteria and  
 return only those matching all of the criteria.  
 We don't need Admins yet, we only filter Users.  
  
Exercise:  
  
 Without duplicating type structures, modify  
 filterUsers function definition so that we can  
 pass only those criteria which are needed,  
 and not the whole User information as it is  
 required now according to typing.  
  
Higher difficulty bonus exercise:  
  
 Exclude "type" from filter criterias.

大概意思是让你改 filterUsers， 但要注意 DRY（Don't Repeat Yourself）。

### 题目内置代码

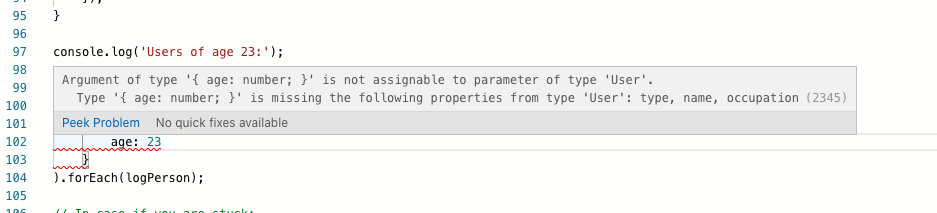
interface User {  
 type: "user";  
 name: string;  
 age: number;  
 occupation: string;  
}  
  
interface Admin {  
 type: "admin";  
 name: string;  
 age: number;  
 role: string;  
}  
  
export type Person = User | Admin;  
  
export const persons: Person[] = [  
 {  
 type: "user",  
 name: "Max Mustermann",  
 age: 25,  
 occupation: "Chimney sweep",  
 },  
 {  
 type: "admin",  
 name: "Jane Doe",  
 age: 32,  
 role: "Administrator",  
 },  
 {  
 type: "user",  
 name: "Kate Müller",  
 age: 23,  
 occupation: "Astronaut",  
 },  
 {  
 type: "admin",  
 name: "Bruce Willis",  
 age: 64,  
 role: "World saver",  
 },  
 {  
 type: "user",  
 name: "Wilson",  
 age: 23,  
 occupation: "Ball",  
 },  
 {  
 type: "admin",  
 name: "Agent Smith",  
 age: 23,  
 role: "Administrator",  
 },  
];  
  
export const isAdmin = (person: Person): person is Admin =>  
 person.type === "admin";  
export const isUser = (person: Person): person is User =>  
 person.type === "user";  
  
export function logPerson(person: Person) {  
 let additionalInformation = "";  
 if (isAdmin(person)) {  
 additionalInformation = person.role;  
 }  
 if (isUser(person)) {  
 additionalInformation = person.occupation;  
 }  
 console.log(` - ${person.name}, ${person.age}, ${additionalInformation}`);  
}  
  
export function filterUsers(persons: Person[], criteria: User): User[] {  
 return persons.filter(isUser).filter((user) => {  
 const criteriaKeys = Object.keys(criteria) as (keyof User)[];  
 return criteriaKeys.every((fieldName) => {  
 return user[fieldName] === criteria[fieldName];  
 });  
 });  
}  
  
console.log("Users of age 23:");  
  
filterUsers(persons, {  
 age: 23,  
}).forEach(logPerson);

### 前置知识

* 泛型
* Partial 泛型

### 思路

老规矩， 从报错入手。



大概意思是 { age: 23 } 不完整，缺失了部分 key。而题目实际上的想法应该是想根据部分内容对人员进行检错。比如可以根据 age 查， 也可以根据 name 查，也可以同时根据 age 和 name 查等，这和我们平时的搜索逻辑是一致的。

直接用 Partial 泛型即可解决， 不懂的可以看下我的文章[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/)。

### 代码

export function filterUsers(persons: Person[], criteria: Partial<User>): User[] {  
 ...  
}

## 第六题

### 题目描述

Intro:  
  
 Filtering requirements have grown. We need to be  
 able to filter any kind of Persons.  
  
Exercise:  
  
 Fix typing for the filterPersons so that it can filter users  
 and return User[] when personType='user' and return Admin[]  
 when personType='admin'. Also filterPersons should accept  
 partial User/Admin type according to the personType.  
 `criteria` argument should behave according to the  
 `personType` argument value. `type` field is not allowed in  
 the `criteria` field.  
  
Higher difficulty bonus exercise:  
  
 Implement a function `getObjectKeys()` which returns more  
 convenient result for any argument given, so that you don't  
 need to cast it.  
  
 let criteriaKeys = Object.keys(criteria) as (keyof User)[];  
 -->  
 let criteriaKeys = getObjectKeys(criteria);

大概意思是让你改 filterUsers， 但要注意 DRY（Don't Repeat Yourself）。并且可以根据 personType 的不同，返回不同的类型。

### 题目内置代码

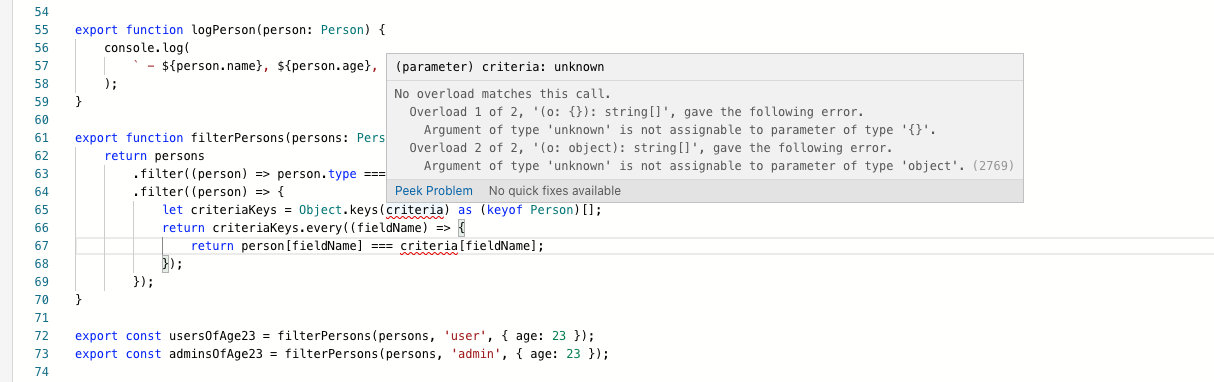
interface User {  
 type: "user";  
 name: string;  
 age: number;  
 occupation: string;  
}  
  
interface Admin {  
 type: "admin";  
 name: string;  
 age: number;  
 role: string;  
}  
  
export type Person = User | Admin;  
  
export const persons: Person[] = [  
 {  
 type: "user",  
 name: "Max Mustermann",  
 age: 25,  
 occupation: "Chimney sweep",  
 },  
 { type: "admin", name: "Jane Doe", age: 32, role: "Administrator" },  
 { type: "user", name: "Kate Müller", age: 23, occupation: "Astronaut" },  
 { type: "admin", name: "Bruce Willis", age: 64, role: "World saver" },  
 { type: "user", name: "Wilson", age: 23, occupation: "Ball" },  
 { type: "admin", name: "Agent Smith", age: 23, role: "Anti-virus engineer" },  
];  
  
export function logPerson(person: Person) {  
 console.log(  
 ` - ${person.name}, ${person.age}, ${  
 person.type === "admin" ? person.role : person.occupation  
 }`  
 );  
}  
  
export function filterPersons(  
 persons: Person[],  
 personType: string,  
 criteria: unknown  
): unknown[] {  
 return persons  
 .filter((person) => person.type === personType)  
 .filter((person) => {  
 let criteriaKeys = Object.keys(criteria) as (keyof Person)[];  
 return criteriaKeys.every((fieldName) => {  
 return person[fieldName] === criteria[fieldName];  
 });  
 });  
}  
  
export const usersOfAge23 = filterPersons(persons, "user", { age: 23 });  
export const adminsOfAge23 = filterPersons(persons, "admin", { age: 23 });  
  
console.log("Users of age 23:");  
usersOfAge23.forEach(logPerson);  
  
console.log();  
  
console.log("Admins of age 23:");  
adminsOfAge23.forEach(logPerson);

### 前置知识

* 泛型
* Partial 泛型
* 函数重载

### 思路

题目描述也懒得看了， 直接看报错。



报错信息提示我们没有找到合适的函数重载。 因此我的思路就是补上合适的重载即可。关于函数重载，我的系列教程不涉及，大家可以看下官网资料。

重载之后，不同的情况调用返回值就可以对应不同的类型。本题中就是：

* 如果 personType 是 admin，就会返回 Admin 数组。
* 如果 personType 是 user，就会返回 User 数组。
* 如果 personType 是其他 string，就会返回 Person 数组。

### 代码

export function filterPersons(persons: Person[], personType: 'admin', criteria: Partial<Person>): Admin[]  
export function filterPersons(persons: Person[], personType: 'user', criteria: Partial<Person>): User[]  
export function filterPersons(persons: Person[], personType: string, criteria: Partial<Person>): Person[] {  
 ...  
}

## 第七题

### 题目描述

Intro:  
  
 Filtering was completely removed from the project.  
 It turned out that this feature was just not needed  
 for the end-user and we spent a lot of time just because  
 our office manager told us to do so. Next time we should  
 instead listen to the product management.  
  
 Anyway we have a new plan. CEO's friend Nick told us  
 that if we randomly swap user names from time to time  
 in the community, it would be very funny and the project  
 would definitely succeed!  
  
Exercise:  
  
 Implement swap which receives 2 persons and returns them in  
 the reverse order. The function itself is already  
 there, actually. We just need to provide it with proper types.  
 Also this function shouldn't necessarily be limited to just  
 Person types, lets type it so that it works with any two types  
 specified.

题目大概意思是让你修改 swap 函数，使得不报错。 并且，我希望这个函数可以适用于任意两个变量，不管其类型一样不一样， 也不管二者类型是什么。

### 题目内置代码

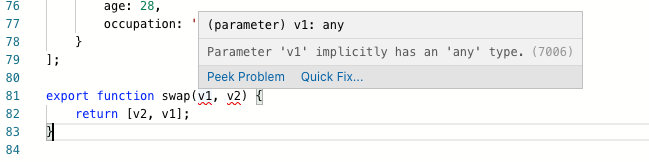
interface User {  
 type: "user";  
 name: string;  
 age: number;  
 occupation: string;  
}  
  
interface Admin {  
 type: "admin";  
 name: string;  
 age: number;  
 role: string;  
}  
  
function logUser(user: User) {  
 const pos = users.indexOf(user) + 1;  
 console.log(` - #${pos} User: ${user.name}, ${user.age}, ${user.occupation}`);  
}  
  
function logAdmin(admin: Admin) {  
 const pos = admins.indexOf(admin) + 1;  
 console.log(` - #${pos} Admin: ${admin.name}, ${admin.age}, ${admin.role}`);  
}  
  
const admins: Admin[] = [  
 {  
 type: "admin",  
 name: "Will Bruces",  
 age: 30,  
 role: "Overseer",  
 },  
 {  
 type: "admin",  
 name: "Steve",  
 age: 40,  
 role: "Steve",  
 },  
];  
  
const users: User[] = [  
 {  
 type: "user",  
 name: "Moses",  
 age: 70,  
 occupation: "Desert guide",  
 },  
 {  
 type: "user",  
 name: "Superman",  
 age: 28,  
 occupation: "Ordinary person",  
 },  
];  
  
export function swap(v1, v2) {  
 return [v2, v1];  
}  
  
function test1() {  
 console.log("test1:");  
 const [secondUser, firstAdmin] = swap(admins[0], users[1]);  
 logUser(secondUser);  
 logAdmin(firstAdmin);  
}  
  
function test2() {  
 console.log("test2:");  
 const [secondAdmin, firstUser] = swap(users[0], admins[1]);  
 logAdmin(secondAdmin);  
 logUser(firstUser);  
}  
  
function test3() {  
 console.log("test3:");  
 const [secondUser, firstUser] = swap(users[0], users[1]);  
 logUser(secondUser);  
 logUser(firstUser);  
}  
  
function test4() {  
 console.log("test4:");  
 const [firstAdmin, secondAdmin] = swap(admins[1], admins[0]);  
 logAdmin(firstAdmin);  
 logAdmin(secondAdmin);  
}  
  
function test5() {  
 console.log("test5:");  
 const [stringValue, numericValue] = swap(123, "Hello World");  
 console.log(` - String: ${stringValue}`);  
 console.log(` - Numeric: ${numericValue}`);  
}  
  
[test1, test2, test3, test4, test5].forEach((test) => test());

### 前置知识

* 泛型

### 思路

题目废话很多， 直接忽略看报错。



这个其实我在 [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/) 里也讲过了，直接看代码。

### 代码

export function swap<U, T>(v1: T, v2: U): [U, T] {  
 return [v2, v1];  
}

## 第八题

### 题目描述

Intro:  
  
 Project grew and we ended up in a situation with  
 some users starting to have more influence.  
 Therefore, we decided to create a new person type  
 called PowerUser which is supposed to combine  
 everything User and Admin have.  
  
Exercise:  
  
 Define type PowerUser which should have all fields  
 from both User and Admin (except for type),  
 and also have type 'powerUser' without duplicating  
 all the fields in the code.

题目大概意思是定义一个类型 PowerUser， 里面包含 User 和 Admin 的所有属性， 并且有一个字段是固定的 type: 'powerUser'。

### 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 PowerUser = unknown;  
  
export type Person = User | Admin | PowerUser;  
  
export const persons: Person[] = [  
 {  
 type: "user",  
 name: "Max Mustermann",  
 age: 25,  
 occupation: "Chimney sweep",  
 },  
 { type: "admin", name: "Jane Doe", age: 32, role: "Administrator" },  
 { type: "user", name: "Kate Müller", age: 23, occupation: "Astronaut" },  
 { type: "admin", name: "Bruce Willis", age: 64, role: "World saver" },  
 {  
 type: "powerUser",  
 name: "Nikki Stone",  
 age: 45,  
 role: "Moderator",  
 occupation: "Cat groomer",  
 },  
];  
  
function isAdmin(person: Person): person is Admin {  
 return person.type === "admin";  
}  
  
function isUser(person: Person): person is User {  
 return person.type === "user";  
}  
  
function isPowerUser(person: Person): person is PowerUser {  
 return person.type === "powerUser";  
}  
  
export function logPerson(person: Person) {  
 let additionalInformation: string = "";  
 if (isAdmin(person)) {  
 additionalInformation = person.role;  
 }  
 if (isUser(person)) {  
 additionalInformation = person.occupation;  
 }  
 if (isPowerUser(person)) {  
 additionalInformation = `${person.role}, ${person.occupation}`;  
 }  
 console.log(`${person.name}, ${person.age}, ${additionalInformation}`);  
}  
  
console.log("Admins:");  
persons.filter(isAdmin).forEach(logPerson);  
  
console.log();  
  
console.log("Users:");  
persons.filter(isUser).forEach(logPerson);  
  
console.log();  
  
console.log("Power users:");  
persons.filter(isPowerUser).forEach(logPerson);

### 前置知识

* 集合操作（交叉类型）
* & 操作符
* 泛型
* Omit 泛型

### 思路

从题目信息不难看出，就是让我们实现 PowerUser。

有前面的分析不难得出我们只需要：

* 合并 User 和 Admin 的属性即可。 借助 & 操作符可以实现。即 User & Admin。
* 增加特有的属性 type: powerUser。 首先去掉上一步合并的 type 属性， 然后继续和 { type: "powerUser" } 交叉即可。
* 增加 { type: "powerUser" } 之前使用内置泛型 Omit 将原本的 type 删掉即可。

### 代码

type PowerUser = Omit<User & Admin, "type"> & { type: "powerUser" };

## 总结

以上就是给大家带来的题目解析。 这八道题的考点有，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）
* 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）
* 函数重载（推荐掌握）

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！