10 接口的高阶用法

更新时间：2019-07-01 14:22:49

> 受苦的人，没有悲观的权利。
>
> ——尼采

学习了上个小节接口的基础用法后，相信你已经能够使用接口来描述一些结构了。本小节我们来继续学习接口，学习接口的高阶用法。接口有一小部分知识与类的知识相关，所以我们放在讲解类的小节后面补充讲解，我们先来学习除了这一小部分之外剩下的接口的知识。

2.7.1 索引类型

我们可以使用接口描述索引的类型和通过索引得到的值的类型，比如一个数组`['a', 'b']`，数字索引`0`对应的通过索引得到的值为`'a'`。我们可以同时给索引和值都设置类型，看下面的示例：

```
interface RoleDic {
  [id: number]: string;
}
const role1: RoleDic = {
  0: "super_admin",
  1: "admin"
};
const role2: RoleDic = {
  s: "super_admin",  // error 不能将类型"{ s: string; a: string; }"分配给类型"RoleDic"。
  a: "admin"
};
const role3: RoleDic = ["super_admin", "admin"];
```

上面的例子中 role3 定义了一个数组，索引为数值类型，值为字符串类型。

你也可以给索引设置`readonly`，从而防止索引返回值被修改。

```
interface RoleDic {
  readonly [id: number]: string;
}
const role: RoleDic = {
  0: "super_admin"
};
role[0] = "admin"; // error 类型"RoleDic"中的索引签名仅允许读取
```

这里有的点需要注意，你可以设置索引类型为 number。但是这样如果你将属性名设置为字符串类型，则会报错；但是如果你设置索引类型为字符串类型，那么即便你的属性名设置的是数值类型，也没问题。因为 JS 在访问属性值的时候，如果属性名是数值类型，会先将数值类型转为字符串，然后再去访问。你可以看下这个例子：

```
const obj = {
  123: "a", // 这里定义一个数值类型的123这个属性
  "123": "b" // 这里在定义一个字符串类型的123这个属性，这里会报错：标识符“"123"”重复。
};
console.log(obj); // { '123': 'b' }
```

如果数值类型的属性名不会转为字符串类型，那么这里数值123和字符串123是不同的两个值，则最后对象obj应该同时有这两个属性；但是实际打印出来的obj只有一个属性，属性名为字符串"123"，而且值为"b"，说明数值类型属性名123被覆盖掉了，就是因为它被转为了字符串类型属性名"123"；又因为一个对象中多个相同属性名的属性，定义在后面的会覆盖前面的，所以结果就是obj只保留了后面定义的属性值。

2.7.2.继承接口

接口可以继承，这和类(类的相关知识，我们会在后面全面详细的学习)一样，这提高了接口的可复用性。来看一个场景：

我们定义一个 Vegetables 接口，它会对 color 属性进行限制。再定义两个接口，一个为 Tomato，一个为 Carrot，这两个类都需要对 color 进行限制，而各自又有各自独有的属性限制，我们可以这样定义：

```
interface Vegetables {
  color: string;
}
interface Tomato {
  color: string;
  radius: number;
}
interface Carrot {
  color: string;
  length: number;
}
```

三个接口中都有对 color 的定义，但是这样写很繁琐，所以我们可以用继承来改写：

```
interface Vegetables {
  color: string;
}
interface Tomato extends Vegetables {
  radius: number;
}
interface Carrot extends Vegetables {
  length: number;
}
const tomato: Tomato = {
  radius: 1.2 // error  Property 'color' is missing in type '{ radius: number; }'
};
const carrot: Carrot = {
  color: "orange",
  length: 20
};
```

上面定义的 tomato 变量因为缺少了从 Vegetables 接口继承来的 color 属性，从而报错。

一个接口可以被多个接口继承，同样，一个接口也可以继承多个接口，多个接口用逗号隔开。比如我们再定义一个 Food 接口， Tomato 也可以继承 Food ：

```
interface Vegetables {
  color: string;
}
interface Food {
  type: string;
}
interface Tomato extends Food, Vegetables {
  radius: number;
}

const tomato: Tomato = {
  type: "vegetables",
  color: "red",
  radius: 1.2
}; // 在定义tomato变量时将继承过来的color和type属性同时声明
```

2.7.3.混合类型接口

JS 的类型是灵活的。在 JS 中，函数是对象类型。对象可以有属性，所以有时我们的一个对象，它既是一个函数，也包含一些属性。比如我们要实现一个计数器函数，比较直接的做法是定义一个函数和一个全局变量：

```
let count = 0;
const countUp = () => count++;
```

但是这种方法需要在函数外面定义一个变量，更优一点的方法是使用闭包：

```
// javascript
const countUp = (() => {
  let count = 0;
  return () => {
    return ++count;
  };
})();
console.log(countUp()); // 1
console.log(countUp()); // 2
```

在 TypeScript3.1 版本之前，我们需要借助命名空间来实现。但是在 3.1 版本，TypeScript 支持直接给函数添加属性，虽然这在 JS 中早就支持了：

```
// javascript
let countUp = () => {
  return ++countUp.count;
};
countUp.count = 0;
console.log(countUp()); // 1
console.log(countUp()); // 2
```

我们可以看到，我们把一个函数赋值给 countUp ，又给它绑定了一个属性 count ，我们的计数保存在这个 count 属性中。

我们可以使用混合类型接口来指定上面例子中 countUp 的类型：

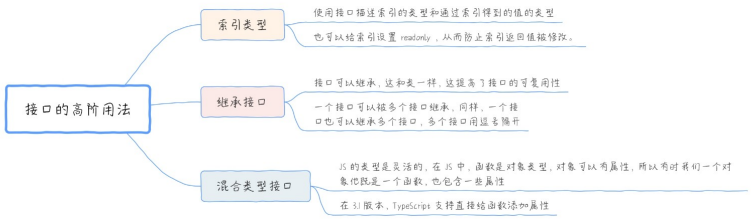
```ts
interface Counter {
  (): void; // 这里定义Counter这个结构必须包含一个函数，函数的要求是无参数，返回值为void，即无返回值
  count: number; // 而且这个结构还必须包含一个名为count、值的类型为number类型的属性
}
const getCounter = (): Counter => { // 这里定义一个函数用来返回这个计数器
  const c = () => { // 定义一个函数，逻辑和前面例子的一样
    c.count++;
  };
  c.count = 0; // 再给这个函数添加一个count属性初始值为0
  return c; // 最后返回这个函数对象
};
const counter: Counter = getCounter(); // 通过getCounter函数得到这个计数器
counter();
console.log(counter.count); // 1
counter();
console.log(counter.count); // 2
```

上面的例子中，`getCounter`函数返回值类型为`Counter`，它是一个函数，无返回值，即返回值类型为`void`，它还包含一个属性`count`，属性返回值类型为`number`。

小结

本小节我们在接口基础知识的基础上，学习了接口的高阶用法。我们学习了如何限定索引的类型，即使用`[]`将索引名括起来，然后使用`: type`来指定索引的类型；还学习了一种复用现有接口的接口定义方式，即继承，使用`extends`关键字实现继承；最后我们通过计数器的例子，学习了如何使用混合类型接口实现更复杂的数据结构。还有一些涉及到类的关于接口的知识，我们会在讲了类之后做一个补充。

下个小节我们将学习函数的相关内容。函数是代码里的重头戏，而且内容较多，我们会分两个小节来讲解，跟紧别掉队哈。



09 使用接口定义几乎任意结构

11 为函数和函数参数定义类型

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示

目前暂无任何讨论