typescript

类型

Record

Map

自定义类型

typescript 中如何定义type

在 TypeScript 中，可以使用 `type` 关键字来定义类型别名和自定义类型。

1. 类型别名

类型别名可以用来定义一个已经存在的类型的别名，方便后续使用。定义方式如下：

```
type MyType = string | number;
```

上面的代码定义了一个类型别名 `MyType`，它是一个 `string` 或 `number` 类型。这个类型别名可以在后续的代码中使用，就像使用 `string` 或 `number` 一样。

1. 自定义类型

除了类型别名，还可以使用 `interface` 关键字来定义自定义类型。自定义类型可以定义一个对象的属性和方法，以及它们的类型。定义方式如下：

```
interface MyInterface {
  name: string;
  age: number;
  sayHello: () => void;
}
```

上面的代码定义了一个自定义类型 `MyInterface`，它有一个 `name` 属性和一个 `age` 属性，它们的类型分别是 `string` 和 `number`；还有一个 `sayHello` 方法，它没有参数，返回类型为 `void`。这个自定义类型可以在后续的代码中使用，就像使用其他类型一样。

typescript中interface的作用

在 TypeScript 中，`interface` 是一种用来定义对象的类型的语法结构。它用于描述一个对象的属性、方法及其类型，以及一些特殊的类型语法结构。使用 `interface` 可以提高代码的可读性、可维护性和可重用性。

下面是一些 `interface` 的作用：

1. 定义对象的类型

使用 `interface` 可以定义一个对象的类型，包括它的属性和方法及其类型。例如：

```
interface Person {
  name: string;
  age: number;
  sayHello: () => void;
}
```

上面的代码定义了一个 Person 类型，它有一个 name 属性和一个 age 属性，它们的类型分别是 string 和 number；还有一个 sayHello 方法，它没有参数，返回类型为 void。

1. 接口继承

interface 可以继承其他 interface，从而扩展已有的类型。例如：

```
interface Student extends Person {
  studentId: string;
}
```

上面的代码定义了一个 Student 类型，它继承了 Person 类型，同时又添加了一个 studentId 属性，它的类型是 string。

1. 用于函数类型

interface 也可以用于定义函数类型。例如：

```
interface AddFunc {
  (a: number, b: number): number;
}
```

上面的代码定义了一个 AddFunc 类型，它是一个函数类型，接收两个参数，都是 number 类型，返回值是一个 number 类型。

1. 可选属性和只读属性

在 interface 中可以定义可选属性和只读属性。例如：

```
interface Person {
  name: string;
  age?: number;
  readonly id: number;
}
```

上面的代码定义了一个 Person 类型，它有一个必选的 name 属性，一个可选的 age 属性，以及一个只读的 id 属性。这意味着 id 属性的值在对象创建后不能被修改。

总之，interface 是 TypeScript 中定义对象类型的一种语法结构，它可以提高代码的可读性、可维护性和可重用性。

typescript中用于函数类型的interface定义后如何使用

在 TypeScript 中，可以使用 interface 来定义函数类型，这个函数类型可以用于声明函数、变量、类等的类型。下面是一个示例：

```
interface AddFunc {
  (a: number, b: number): number;
}

const add: AddFunc = (a: number, b: number) => a + b;
```

上面的代码中，AddFunc 定义了一个函数类型，它接收两个参数，都是 number 类型，返回值是一个 number 类型。然后定义了一个函数 add，它的类型是 AddFunc，即符合 AddFunc 定义的函数类型，可以使用它来实现加法操作。

在 TypeScript 中，定义函数类型的接口可以让我们更加明确地描述函数的参数类型和返回值类型，从而可以提高代码的可读性和可维护性。同时，还可以利用这个接口来检查函数的类型是否符合要求，从而避免一些类型错误的问题。

另外，在定义接口时，可以使用可选参数和默认参数来定义函数类型。例如：

```
interface Greeter {
  (name: string, greeting?: string): string;
}

const greet: Greeter = (name: string, greeting: string = 'Hello') => `${greeting}, ${name}!`;
```

上面的代码中，`Greeter` 定义了一个函数类型，它接收一个必选参数 `name` 和一个可选参数 `greeting`，返回值是一个 `string` 类型。然后定义了一个函数 `greet`，它的类型是 `Greeter`，可以使用它来实现问候语。在调用 `greet` 函数时，如果没有传入 `greeting` 参数，则默认使用 `'Hello'`。

总之，使用接口定义函数类型可以让我们更好地描述函数的参数和返回值，提高代码的可读性和可维护性，同时还可以利用它来检查函数类型是否符合要求。

typescript中使用class声明的对象 如何导出成一个类型

在 TypeScript 中，可以使用 `class` 关键字来定义一个类，通过 `new` 关键字可以实例化一个类并创建一个对象。如果想要导出一个类的类型，可以使用 `type` 关键字来定义一个类型别名。下面是一个示例：

```
export class Person {
  constructor(public name: string, public age: number) {}

  sayHello(): void {
    console.log(`Hello, my name is ${this.name}, I'm ${this.age} years old.`);
  }
}

export type PersonType = Person;
```

上面的代码中，`Person` 类定义了一个人的属性和方法。通过 `type` 关键字定义了一个 `PersonType` 类型别名，类型别名就是 `Person` 类型，这样就可以将 `Person` 类型导出成一个类型供其他模块使用了。

可以将上面的代码保存在一个名为 `person.ts` 的文件中，然后在另一个文件中导入 `PersonType` 类型别名并使用：

```
import { PersonType } from './person';

function greet(person: PersonType): void {
  person.sayHello();
}

const p = new Person('Alice', 20);
greet(p);
```

在上面的代码中，使用 `import` 关键字导入了 `PersonType` 类型别名，然后定义了一个函数 `greet`，它接收一个 `PersonType` 类型的参数，这个参数就是 `Person` 类型，可以调用 `sayHello` 方法。最后创建了一个 `Person` 类型的对象 `p`，并调用了 `greet` 函数。