TypeScript 联合类型 TypeScript 类

TypeScript 接口

接口是一系列抽象方法的声明，是一些方法特征的集合，这些方法都应该是抽象的，需要由具体的类去实现，然后第三方就可以通过这组抽象方法调用，让具体的类执行具体的方法。

TypeScript 接口定义如下：

```
interface interface_name {
}
```

实例

以下实例中，我们定义了一个接口 IPerson，接着定义了一个变量 customer，它的类型是 IPerson。

customer 实现了接口 IPerson 的属性和方法。

TypeScript

```
interface IPerson {
firstName:string,
lastName:string,
sayHi: ()=>string
}
var customer:IPerson = {
firstName:"Tom",
lastName:"Hanks",
sayHi: ():string =>{return "Hi there"}
}
console.log("Customer 对象 ")
console.log(customer.firstName)
console.log(customer.lastName)
console.log(customer.sayHi())
var employee:IPerson = {
firstName:"Jim",
lastName:"Blakes",
sayHi: ():string =>{return "Hello!!!"}
}
console.log("Employee 对象 ")
console.log(employee.firstName)
console.log(employee.lastName)
```

需要注意接口不能转换为 JavaScript。 它只是 TypeScript 的一部分。

编译以上代码，得到以下 JavaScript 代码：

JavaScript

```
var customer = {
firstName: "Tom",
lastName: "Hanks",
sayHi: function () { return "Hi there"; }
```

```
};
console.log("Customer 对象 ");
console.log(customer.firstName);
console.log(customer.lastName);
console.log(customer.sayHi());
var employee = {
firstName: "Jim",
lastName: "Blakes",
sayHi: function () { return "Hello!!!"; }
};
console.log("Employee 对象 ");
console.log(employee.firstName);
console.log(employee.lastName);
```

输出结果为：

```
Customer 对象
Tom
Hanks
Hi there
Employee  对象
Jim
Blakes
```

联合类型和接口

以下实例演示了如何在接口中使用联合类型：

TypeScript

```
interface RunOptions {
program:string;
commandline:string[]|string|(()=>string);
}
// commandline 是字符串
var options:RunOptions = {program:"test1",commandline:"Hello"};
console.log(options.commandline)
// commandline 是字符串数组
options = {program:"test1",commandline:["Hello","World"]};
console.log(options.commandline[0]);
console.log(options.commandline[1]);
// commandline 是一个函数表达式
options = {program:"test1",commandline:()=>{return "**Hello World**";}};
var fn:any = options.commandline;
console.log(fn());
```

编译以上代码，得到以下 JavaScript 代码：

JavaScript

```
// commandline 是字符串
var options = { program: "test1", commandline: "Hello" };
```

```
console.log(options.commandline);
// commandline 是字符串数组
options = { program: "test1", commandline: ["Hello", "World"] };
console.log(options.commandline[0]);
console.log(options.commandline[1]);
// commandline 是一个函数表达式
options = { program: "test1", commandline: function () { return "**Hello World**"; } };
var fn = options.commandline;
console.log(fn());
```

输出结果为：

```
Hello
Hello
World
**Hello World**
```

接口和数组

接口中我们可以将数组的索引值和元素设置为不同类型，索引值可以是数字或字符串。

TypeScript

```
interface namelist {
[index:number]:string
}
var list2:namelist = ["John",1,"Bran"] / 错误元素 1 不是 string 类型
interface ages {
[index:string]:number
}
var agelist:ages;
agelist["John"] = 15 // 正确
agelist[2] = "nine" // 错误
```

接口继承

接口继承就是说接口可以通过其他接口来扩展自己。

Typescript 允许接口继承多个接口。

继承使用关键字 `extends`。

单接口继承语法格式：

```
Child_interface_name extends super_interface_name
```

多接口继承语法格式：

```
Child_interface_name extends super_interface1_name, super_interface2_name,…,super_interfaceN_name
```

继承的各个接口使用逗号 `,` 分隔。

单继承实例

TypeScript

```
interface Person {
age:number
}
interface Musician extends Person {
instrument:string
}
var drummer = <Musician>{};
drummer.age = 27
drummer.instrument = "Drums"
console.log("年龄: "+drummer.age)
console.log("喜欢的乐器: "+drummer.instrument)
```

编译以上代码，得到以下 JavaScript 代码：

JavaScript

```
var drummer = {};
drummer.age = 27;
drummer.instrument = "Drums";
console.log("年龄: " + drummer.age);
console.log("喜欢的乐器: " + drummer.instrument);
```

输出结果为：

```
年龄:  27
喜欢的乐器:  Drums
```

多继承实例

TypeScript

```
interface IParent1 {
v1:number
}
interface IParent2 {
v2:number
}
interface Child extends IParent1, IParent2 { }
var Iobj:Child = { v1:12, v2:23}
console.log("value 1: "+Iobj.v1+" value 2: "+Iobj.v2)
```

编译以上代码，得到以下 JavaScript 代码：

JavaScript

```
var Iobj = { v1: 12, v2: 23 };
console.log("value 1: " + Iobj.v1 + " value 2: " + Iobj.v2);
```

输出结果为：

```
value 1: 12 value 2: 23
```

TypeScript 联合类型 TypeScript 类