使用模块封装代码

更新时间：2019-07-09 16:32:59

> 横眉冷对千夫指，俯首甘为孺子牛。
>
> ——鲁迅

TypeScript 在 1.5 版本之前，有**内部模块**和**外部模块**的概念，从 1.5 版本开始，**内部模块**改称作**命名空间**（我们下个小节会讲），**外部模块**改称为**模块**。如果你对模块的知识一无所知，建议你先重点学习一下 `CommonJS` 模块系统和 ES6模块系统，TypeScript 中的模块系统是遵循 ES6 标准的，所以你需要重点学习 ES6 标准中的模块知识，这里推荐大家几个链接，大家可以在这里去学习一下：

- CommonJS/AMD/CMD/ES6规范
- ECMAScript6入门 - Module 的语法

TypeScript 和 ES6 保持一致，包含顶级 import 或 export 的文件都被当成一个模块，则里面定义的内容仅模块内可见，而不是全局可见。TypeScript 的模块除了遵循 ES6 标准的模块语法外，还有一些特定语法，用于类型系统兼容多个模块格式，下面我们来开始学习 TypeScript 模块。

4.1.1. export

TypeScript 中，仍然使用 export 来导出声明，而且能够导出的不仅有变量、函数、类，还包括 TypeScript 特有的类型别名和接口。

```
// funcInterface.ts
export interface Func {
  (arg: number): string;
}
export class C {
  constructor() {}
}
class B {}
export { B };
export { B as ClassB };
```

上面例子中，你可以使用 export 直接导出一个声明，也可以先声明一个类或者其它内容，然后使用 export {}的形式导出，也可以使用 as 来为导出的接口换个名字再导出一次。

你也可以像 ES6 模块那样重新导出一个模块，也就是 export 一个引入内容，也可以重新导出部分内容，也可以重命名重新导出：

```
// main.ts
export * from "./moduleB";
// main.ts
export { name } from "./moduleB";
// main.ts
export { name as nameProp } from "./moduleB";
```

4.1.2. import

接下来我们来看导出的模块怎么引入，依然是使用 import：

```
// main.ts
import { name } from "./moduleB";
// main.ts
import * as info from "./moduleB";
//main.ts
import { name as nameProp } from "./moduleB";
```

同样，可以使用 import 直接接模块名或文件路径，进行具有副作用的导入：

```
import "./set-title.ts";
```

4.1.3. export default

同样在 TypeScript 中使用 export default 默认导出，这个和 ES6 一样：

```
// moduleB.ts
export default "lison";
// main.ts
import name from "./moduleB.ts";
console.log(name); // 'lison'
```

4.1.4. export = 和 import = require()

TypeScript可以将代码编译为CommonJS、AMD或其它模块系统代码，同时会生成对应的声明文件。我们知道CommonJS和AMD两种模块系统语法是不兼容的，所以TypeScript为了兼容这两种语法，使得我们编译后的声明文件同时支持这两种模块系统，增加了`export =`和`import xx = require()`两个语句。

当我们想要导出一个模块时，可以使用`export =`来导出：

```
// moduleC.ts
class C {}
export = C;
```

然后使用这个形式导出的模块，必须使用`import xx = require()`来引入：

```
// main.ts
import ClassC = require("./moduleC");
const c = new ClassC();
```

如果你的模块不需要同时支持这两种模块系统，可以不使用`export =`来导出内容。

4.1.5. 相对和非相对模块导入

根据引入模块的路径是相对还是非相对，模块的导入会以不同的方式解析：

- 相对导入是以`./`或`../`开头的，`./`表示当前目录，而`../`表示当前目录的上一级目录。以下面的文件目录为例：

```
src
 ├─ module
 │    ├─ moduleA.ts
 │    └─ moduleB.ts
 └─ core
      └─ index.ts
```

以上面的文件目录为例，我们如果在 index.ts 中引入两个模块，和在 moduleA 模块中引入 moduleB 是这样的：

```
// moduleA.ts
import moduleB from "./moduleB.ts"; // 这里在moduleA.ts文件里引入同级的moduleB.ts文件，所以使用./表示moduleA.ts文件当前所在路径
// index.ts
import moduleA from "../module/moduleA.ts";
import moduleB from "../module/moduleB"; // 这里省略了.ts后缀也可以
```

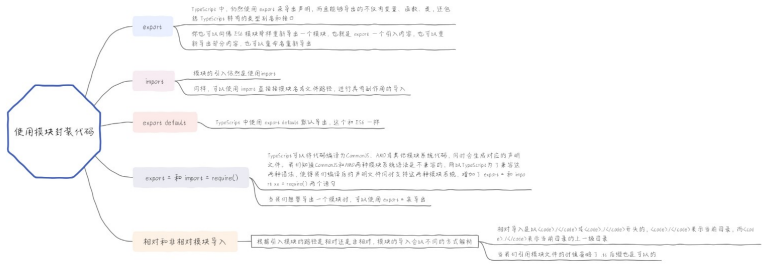
当我们引用模块文件的时候省略了 .ts 后缀也是可以的，这就涉及到一个模块解析策略。我们以上面例子中这个 moduleB 为例，编译器在解析模块引用的时候，如果遇到省略后缀的情况，会依次查找以该名称为文件名的.ts、.tsx、.d.ts文件；如果没找到，会在当前文件夹下的 package.json 文件里查找 types 字段指定的模块路径，然后通过这个路径去查找模块；如果没找到package.json文件或者types字段，则会将 moduleB 当做文件夹去查找，如果它确实是文件夹，将会在这个文件夹下依次查找 index.ts、index.tsx、index.d.ts。 如果还没找到，会在上面例子中 module 文件夹的上级文件夹继续查找，查找规则和前面这些顺序一致。

除了这两种符号开头的路径，都被当做非相对路径。非相对模块的导入可以相对于 baseUrl，也可以通过路径映射，还可以解析为外部模块。关于模块解析的更多配置，我们会在后面章节介绍，这里我们主要学习语法。

小结

本小节我们学习了 TypeScript 的模块系统，使用模块我们可以将一些逻辑封装在模块中，方便在多个文件中引入使用，这样可以帮我们更方便地整理、复用代码。在这个小节中我们学习了 TypeScript 中遵循 ES6 的模块语句，其中导出语句为`export`，引入语句为`import`，如果要默认导出，则使用`export default`。我们还学习了 TypeScript 中特别增加的兼容 CommonJS 和 AMD 模块系统的导出和导入语句：`export =`和`import xx = require()`。最后我们简单学习了相对模块和非相对模块的引入，同时还介绍了模块引用的解析策略。

下个小节我们将学习命名空间，命名空间在TypeScirpt1.5版本之前称为“内部模块”，它可以将属于一类的代码进行汇总封装，效果类似于我们这节课学习的模块，下个小节我们来进行学习。



25 入手装饰器，给凡人添加超能力

使用命名空间封装代码